ПРОГРАММИРОВАНИЕ НА C++

Указания к выполнению упражнений

Необходимо выполнить все задания, указанные в упражнениях и контрольные задания, приведенные в конце практических занятий.

Представить для отчетности код контрольных заданий с необходимыми пояснениями в комментариях.

СОДЕРЖАНИЕ

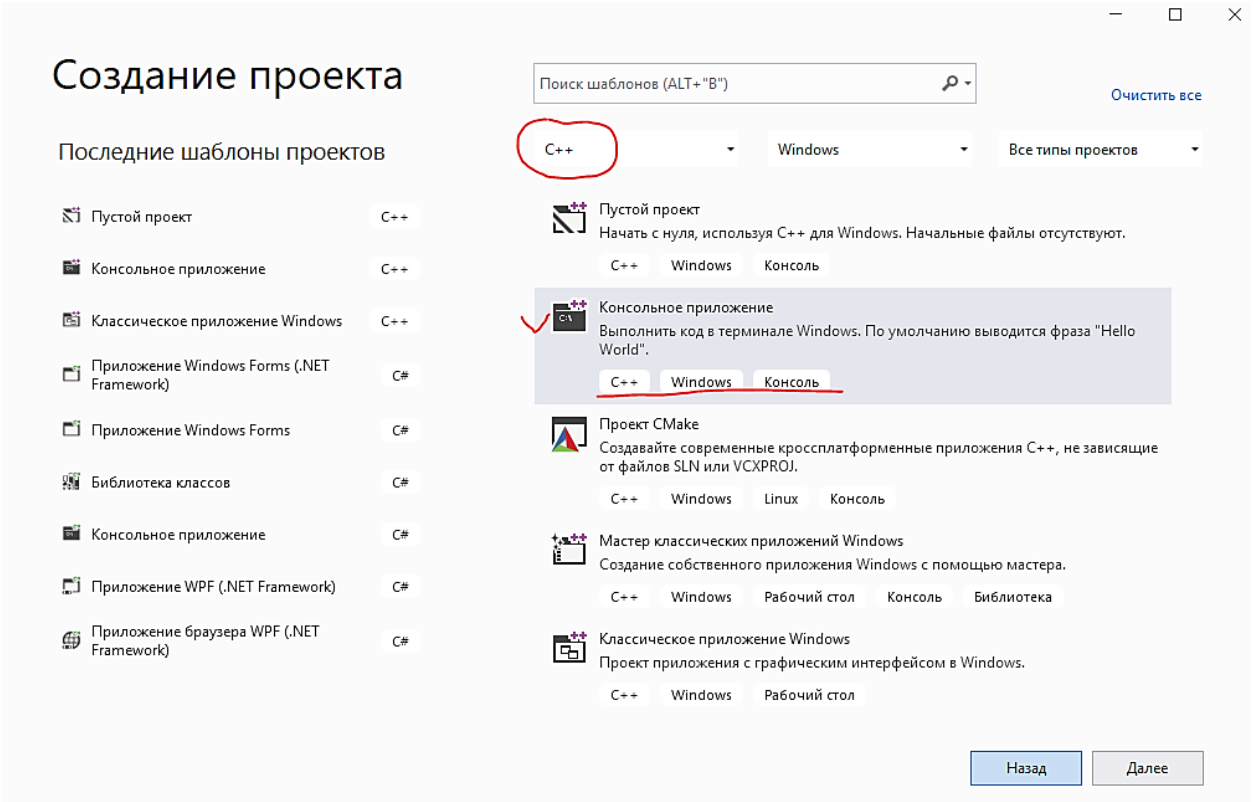
Предварительное занятие. Создание программы на языке C++ при помощи среды разработки

В этих упражнениях вы приобретете навыки создания проекта для разработки программы на языке C++. Выберите один из двух IDE.

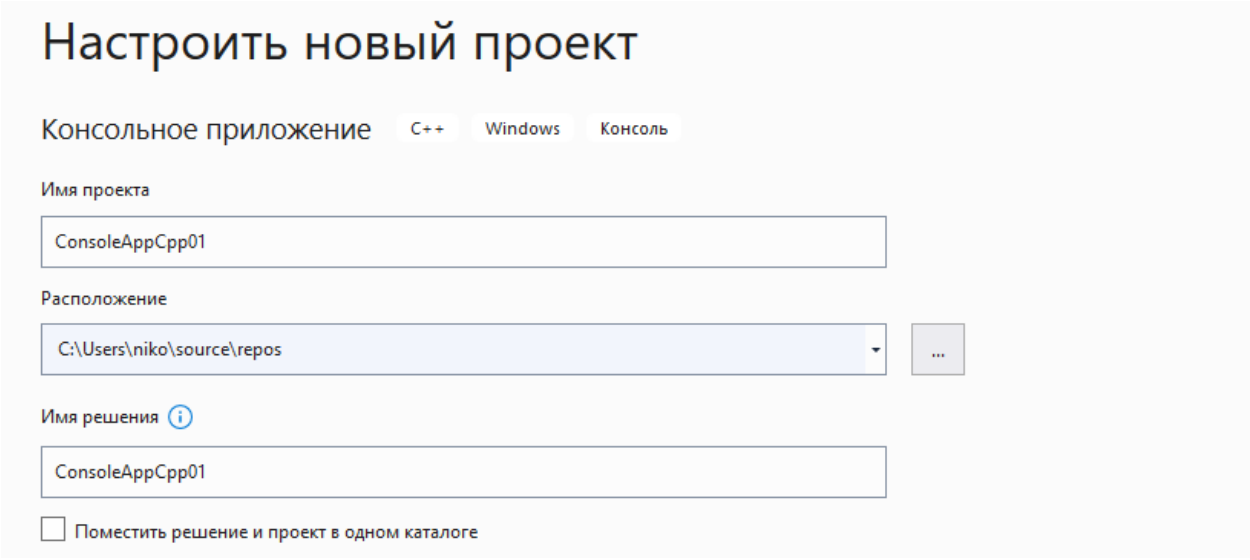
Упражнение 1. Использование интегрированной среды разработки Visual Studio

Запустите среду разработки **Visual Studio.**

1. Первым шагом является создание нового проекта. Для этого выберите **Создание проекта**.
2. Укажите параметры как на рисунке:



3. Укажите имя проекта и выберите расположение и нажмите кнопку **Создать**:



4. Откроется окно проекта.
5. Изучите структуру проекта и содержимое файла с функцией main().
6. Постройте и запустите приложение.
7. Реализуйте русификацию консоли.

Другой вариант русификации консоли

Для русификации консоли может быть использован следующий способ.

1. Файл исходного текста сохраните в формате с кодировкой UTF8.
2. В код добавьте инструкцию

```
#include <windows.h>
```

3. В начале функции `main()` укажите номера кодовых страниц для вывода и ввода русскоязычного текста (или поддержку UTF8 как в данном примере):

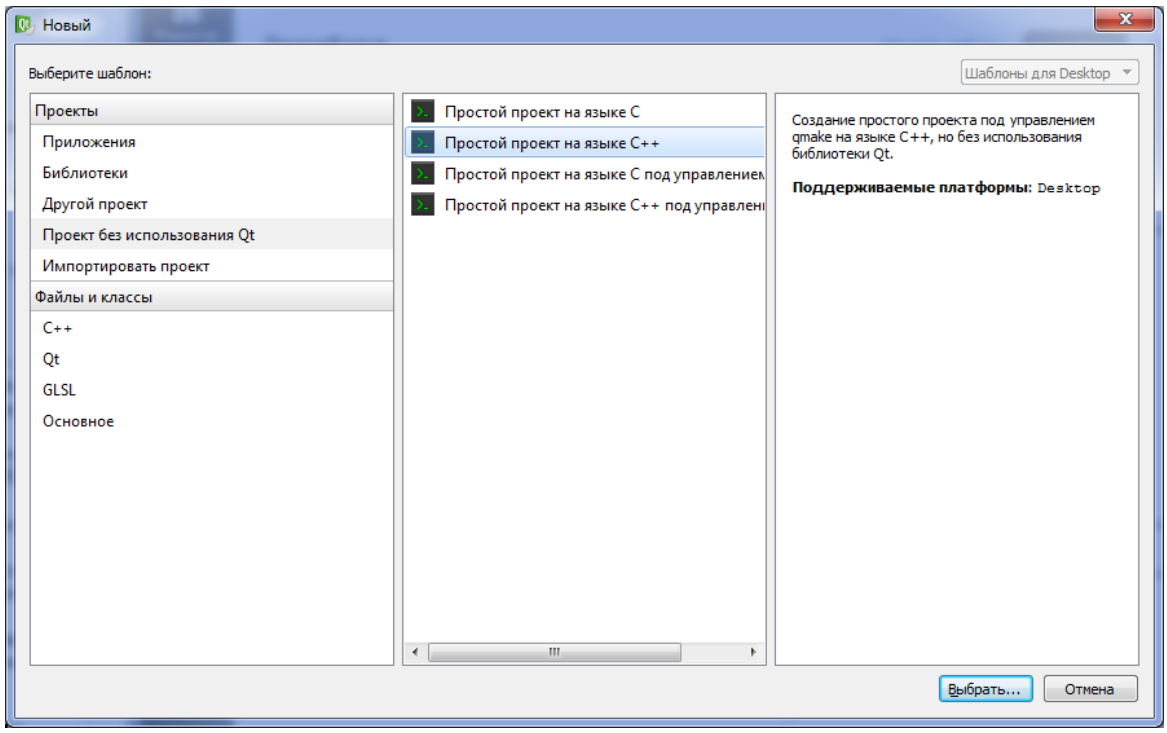
```
SetConsoleOutputCP(CP_UTF8);
SetConsoleCP(CP_UTF8);
```

4. В программе замените строки вывода информации на русский язык.
5. Запустите программу на выполнение.
6. Откройте свойства консоли и на вкладке настройки шрифта укажите шрифт *Lucida Console.*
7. Нажмите кнопку OK. Текст должен отображаться правильно.

Упражнение 2. Использование интегрированной среды разработки Qt Creator

Запустите среду разработки **Qt Creator.**

1. Первым шагом является создание нового проекта. Для этого в меню **Файл** выберите **Новый файл или проект**.
2. В окне **Новый** выберите шаблон **Проект без использования Qt** и далее выберите Простой проект на C++ (см. рис), нажмите кнопку **Выбрать**.



3. В поле **Название** введите имя проекта – **MyFistProg.**
4. В поле **Создать в** проверьте путь размещения проекта – по умолчанию проект сохраняется в специальной папке **QPrim**. Оставьте без изменений (при желании можно выбрать путь с помощью кнопки **Обзор**.
5. Нажмите кнопку **Далее** и в окне **Управление проектом** нажмите **Завершить.**
6. Откроется окно проекта.
7. Изучите структуру проекта и содержимое файла main.cpp:

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

8. С помощью меню **Сборка** запустите приложение на выполнение.
9. Для поддержки кириллицы в функцию `main()` добавьте указание кодовой страницы:

```
system("chcp 1251");
```

10. Замените в строке вывода английский язык на русский и проверьте работу программы.

Практическое занятие 1. Организация ввода-вывода данных

Упражнение 1. Ввод-вывод данных

В этом упражнении изучаются особенности ввода текстовых данных с клавиатуры и вывод их на экран.

Замечание. Если вы будете использовать кириллицу, то не забывайте использовать один из описанных ранее способов.

1. Создайте новый проект и в файле исходного кода (.cpp) введите текст программы:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string name;
  cout << "What is your name? ";
  cin >> name;
  cout << "Hello, " << name << "!\n";
}
```

2. Изучите объекты ввода и вывода данных:

`cin` – стандартный ввод – с клавиатуры,

`cout` – стандартный вывод на экран монитора.

3. Запустите программу на выполнение. Введите имя, состоящее из нескольких слов, заметьте, что сохранилось в переменной только первое слово.
4. Замените строку `cin >> name;` на вызов функции чтения строки – `getline()`:

```
getline(cin, name);
```

5. Снова запустите программу, проверьте, что в программу передается полностью строка – имя и фамилия.

Упражнение 2. Преобразование типов данных

В этом упражнении вы изучите особенности ввода числовых данных с клавиатуры и вывод их на экран, а также преобразование типов данных.

1. Создайте новый проект и в файле исходного кода (.cpp) введите текст программы:

```
#include <iostream>

using namespace std;
```

```
int main()
{
    system("chcp 1251");
    double a, b;
    cout << "Введите a и b:\n";
    cin >> a;     // ввод с клавиатуры значения a
    cin >> b;     // ввод с клавиатуры значения b
    double x = a / b;    // вычисление значения x
cout << "\nx = " << x << endl;  //вывод результата на экран
    return 0;
}
```

2. Запустите программу. Введите значение для переменной `a`, затем нажмите ввод и введите значение для переменной `b`.
3. Проверьте, что выводится правильный результат.
4. Замените объявление переменной `x`: вместо типа `double` укажите тип `int`.
5. Запустите программу. Разделите, например 17 на 3. Должен получиться результат: 5. Произошло неявное преобразование типов.

➢ Результат вычисления дроби будет иметь тип `double`, перед выполнением операции присваивания он преобразуется к типу `int`, который имеет переменная `x`, стоящая в левой части оператора, таким образом, дробная часть результата отброшена.

6. Добавьте в функцию `main()` вызов функции `sizeof()` для расчета объема занимаемой памяти:

```
cout << sizeof(a/b) << ends << sizeof(x)<< endl;
```

7. Запустите снова программу и сравните объем занимаемой памяти дроби `a/b` и переменной `x`.
8. Верните для переменной `x` правильный тип – `double` и проверьте работу программы.

Форматирование вывода cout

➢ Количество десятичных разрядов, отображаемых справа от точки (точность) по умолчанию, применяемая в C++ равна 6. Функция-член `cout.precision(k)` позволяет указать требуемое значение точности (параметр k). Установка точности остается в действии до тех пор, пока она не будет переустановлена.

9. Измените формат вывода результата: для обеспечения вывода числа с точностью до трех разрядов после запятой, добавьте перед выводом результата (в принципе можно в любое место функции main()) следующую инструкцию:

```
cout.precision(3);
```

10. Повторите запуск программы и проверьте работу добавленной функции.

➢ Более подробно про форматирование вывода.

Исследование ввода операцией cin >>

- Объект **cin** представляет стандартный ввод в виде потока байтов (стандартный поток символов генерируется клавиатурой). При вводе последовательности символов объект **cin** извлекает эти символы из входного потока.
- Поскольку вводом может являться часть строки, значением типа int, float или какого-либо иного типа, то извлечение символов из потока предполагает также преобразование типа. Объект **cin** на основании типа переменной, предназначенной для приема значения, должен применяет свои методы для преобразования последовательности символов в значения соответствующего типа.
- Операция извлечения (**>>**) одинаково воспринимает входной поток. Она опускает пробельные символы (пробелы, символы новой строки и табуляции) до тех пор, пока не встретят не пробельный символ.

11. Запустите снова программу, но теперь введите значения переменных через пробел. Проверьте правильность расчетов.

- Операция (**>>**) считывает один элемент данных указанного типа. То есть она читает все, начиная с начального символа, отличного от пробельного, вплоть до первого символа, не соответствующего целевому типу.

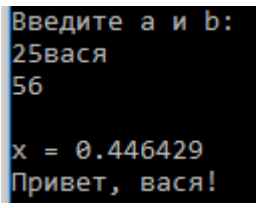
12. Проверьте последнее замечание. Укажите запрос имени после запроса параметра `a` (выделено жирным) и перенесите вывод приветствия в конец функции:

```
int main()
{
    system("chcp 1251");
    string name;
    cout << "Введите свое имя\n";
    double a, b;
    cout << "Введите a и b:\n";
    cin >> a;
    cin >> name;
    cin >> b;
    double x = a / b;
    cout << "\nx = " << x << endl;
    cout << "Привет, " << name << "!\n";
    return 0;
}
```

13. Запустите программу. Введите значение параметра `a` слитно с именем и нажмите ввод:

```
25вася
```

14. Затем введите значение параметра b, например: 7 и нажмите ввод.
15. Проверьте, что операция (>>) считала символы 25 указанного типа и как только встретила другой тип прекратила заполнять переменную `a`. Остальные символы заполнили переменную `name` соответствующего типа:



Упражнение 3. Применение библиотеки fmt

В этом упражнении применяется библиотека **fmt** (также известна как {fmt}), современная, быстрая и удобная альтернатива стандартному printf и std::cout.

1. Создайте новый проект и в файле исходного кода (.cpp) введите текст программы:

```
##include <fmt/core.h>
#include <string>

int main() {
    int x = 42;
    std::string name = "Alice";

    // Печать с форматированием
    fmt::print("Hello, {}, the answer is {}!\n", name, x);

    return 0;
}
```

2. Проверьте, что на экран вывелась информация в виде строки:

```
Hello, Alice, the answer is 42!
```

3. Добавьте и другие варианты форматирования:
   - Чисел в других системах счисления

```
    fmt::print("Hex: {:#x}\n", num);   // шестнадцатеричное с префиксом 0x
```

   - Управление шириной и выравниванием

```
    fmt::print("|{:>10}|\n", 42);  // выравнивание вправо
    fmt::print("|{:<10}|\n", 42);  // выравнивание влево
    fmt::print("|{:^10}|\n", 42);  // выравнивание по центру
```

   - Форматирование с плавающей точкой

```
    fmt::print("Fixed: {:.2f}\n", pi);  // 2 знака после запятой
    fmt::print("Scientific: {:.3e}\n", pi); // экспоненциальный формат
```

   - Использование с std::string (форматирование без вывода)

```
    std::string s = fmt::format("Pi ≈ {:.3f}", 3.14159);
    fmt::print("{}\n", s);
```

4. Изучите объекты вывода данных.
5. Самостоятельно реализуйте следующие указания:

- Возвратите строку с числом в разных форматах: десятичном, шестнадцатеричном и двоичном, как например:

```
"Dec: 42, Hex: 0x2a, Bin: 0b101010"
```

- Строку с числом в трёх форматах:
  - по умолчанию,
  - с 2 знаками после запятой,
  - в экспоненциальной форме

```
"Default: 3.14159 | Fixed: 3.14 | Sci: 3.142e+00"
```

- Создайте сообщение для контроля процесса в следующем виде:
  - метка времени,
  - уровень лога,
  - сообщение.

Формат записи: `[YYYY-MM-DD HH:MM:SS] [LEVEL] message`

Пример вывода:

```
[2025-08-20 14:23:45] [INFO] Application started
[2025-08-20 14:23:45] [WARNING] Low disk space
[2025-08-20 14:23:45] [ERROR] Failed to open file
```

Метку времени можно получить с помощью встроенной функции (подключите `#include <chrono>`):

```
auto now = std::chrono::system_clock::now();
```

Сообщение можно будет потом использовать для логера.

Упражнение 4. Расчет площади треугольника

В этом упражнении требуется написать программу, которая подсчитывает площадь равностороннего треугольника, периметр которого известен.

Реализуйте диалог с пользователем:

– с клавиатуры пользователь вводит значение периметра,
– на экран информация (вещественные значения должны иметь не более двух знаков после запятой) должна выводиться в виде небольшой таблицы:

Сторона	Площадь
Значение	*Результат*

Для расчета площади используйте формулу Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где p – полупериметр.

Функция вычисления квадратного корня (`sqrt`) определена в файле **cmath** и требуется его указать в директиве:

```
#include<cmath>
```

Контрольные задания.

Задание 1. Расчет площади многоугольника

Выпуклый пятиугольник задан на декартовой плоскости координат с помощью координат вершин.

Требуется составить программу для расчета площади данного пятиугольника.

Определите типы данных, используемые в расчетах.

Расчет площади проведите по формуле площади Гаусса (обратите внимание на сферы применения данной формулы), непосредственно использующей координаты вершин.

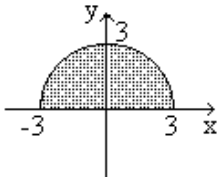
Координаты вершин пользователь должен вводить с клавиатуры.

Практическое занятие 2. Реализация управляющих операторов

Упражнение 1. Реализация операторов выбора

Задание 1. Определение принадлежности точки заданной фигуре

Требуется составить алгоритм и программу для определения принадлежности точки заданной фигуре на плоскости:



Исходные данные: координаты точки – x,y.

Результат: сообщение «внутри», «снаружи», «на границе».

- Проверить попадает ли точка в область фигуры на рисунке можно, используя конструкцию **if-else-if**:

```
if (x * x + y * y < 9 &&y > 0)
       // "внутри"
else if (x * x + y * y> 9 || y< 0)
// "снаружи"
else // "на границе"
```

Задание 2. Использование оператора switch при реализации выбора

В этом упражнении вы используете оператор **switch** при реализации выбора варианта из множества альтернатив.

1. Создайте новый проект и добавьте файл исходного кода.

В методе `main()` объявите переменную символьного типа, которая будет определять выбор пользователя:

```
char op;
```

2. Реализуйте запрос от пользователя для выбора:

```
cout << "Сделай свой выбор, собери авто свой мечты:  ";
cin >> op;
```

3. Реализуйте выбор альтернативного варианта с использованием оператора `switch`:

```
switch (op)
      {
      case 'S':
            cout << "Радио играть должно\n";
            cout << "Колеса круглые\n";
            cout << "Мощный двигатель\n";
            break;

      case 'V':
            cout << "Кондиционер хочу\n";
            cout << "Радио играть должно\n";
            cout << "Колеса круглые\n";
            cout << "Мощный двигатель\n";
            break;

      default:
            cout << "Колеса круглые\n";
            cout << "Мощный двигатель\n";
      }
```

4. Постройте и протестируйте приложение.

➢ В отличие от C# в языке C++ в выражении `switch` не обязательно наличие оператора `break`.

5. Обратите внимание, что в case-выражениях есть дублирование кода. Измените тело оператора `switch`, убрав дублирование. Для решения требуемой задачи удалите в case-выражениях оператор `break` И измените порядок проверки ветвей – используйте проваливание по ветвям для набора функционала.
6. Возможное решение выглядит следующим образом:

```
switch (op)
{
      case 'V':
            cout << "Кондиционер хочу\n";
      case 'S':
            cout << "Радио играть должно\n";
      default:
            cout << "Колеса круглые\n";
            cout << "Мощный двигатель\n";
}
```

Упражнение 2. Использование циклов при реализации алгоритмов

Задание 1. Использование цикла с постусловием

В этом задании вы используете цикл с постусловием для вывода значения функции на интервале.

1. Создайте новый проект и добавьте файл исходного кода.
2. В методе `main()` объявите четыре переменных вещественного типа, `x` – аргумент функции, `x1`, `x2` – границы интервала, `y` – выходной параметр функции, для границ интервала реализуйте ввод значений с клавиатуры:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
     double x, x1, x2,y;
     cout << "x1 = "; cin >> x1;
     cout << "x2 = "; cin >> x2;
```

3. Реализуйте печать заголовка таблицы вывода значений функции:

```
     cout << "\tx\tsin(x)\n";
```

4. С помощью цикла с постусловием реализуйте вывод значений функции `sin(x)` на интервале от `x1` до `x2` с шагом 0,01:

```
     x = x1;
     do
     {
          y = sin(x);
          cout << "\t" << x << "\t" << y << endl;
          x = x + 0.01;
     }
     while (x <= x2);
return 0;
}
```

5. Постройте и протестируйте приложение.

Задание 2. Использование цикла с предусловием

В этом задании вы используете цикл с предусловием для определения значения наибольшего общего делителя двух целых чисел по алгоритму Евклида.

➢ Универсальным способом, позволяющим вычислять наибольший общий делитель (наибольший общий делитель (НОД) – наибольшее число, на которое можно разделить несколько чисел без остатка) двух положительных целых чисел является алгоритм Евклида. Алгоритм Евклида с вычитанием (один из возможных вариантов реализации) заключается в последовательной замене наибольшего числа из двух данных чисел, для которых вычисляется НОД, разностью этих чисел.

1. Создайте новый проект и добавьте файл исходного кода.

2. В функции `main()` объявите три целочисленные переменные, `a` и `b` – исходные данные, `temp` – временная переменная для реализации алгоритма:

```
int a, b, temp;
```

3. Реализуйте ввод значений переменных `a` и `b`:

```
cout << "a = "; cin >> a;
cout << "b = "; cin >> b;
```

4. С помощью цикла с предусловием реализуйте алгоритм Евклида:

```
while (a!=b)
{
      if (a > b)
            a -= b; // аналогично выражению a = a - b
      else
            b -= a;
}
cout << "НОД = " << a << endl;
```

5. Постройте и протестируйте приложение.

Задание 3. Сравнение типов цикла

Реализуйте задачу первого задания с помощью цикла с предусловием, а второго задания с постусловием.

Сравните варианты реализации задач с помощью разных циклов и сделайте выводы о целесообразности выбора типа цикла.

Упражнение 3. Применение цикла с параметром (for) для итерации

➢ Итерация – это способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

Для реализации итерации применяется цикл `for`.

Цикл с параметром применяется в случае, когда известно количество итераций – повторений некоторых действий – тела цикла.

Требуется составить программу тестирования знаний таблицы умножения. Тест должен иметь заранее известное число вопросов и подсчитывать количество неправильных ответов.

Для генерации случайных множителей будет использована функция `rand`(), а для привязки процесса генерации к системному времени (чтобы при повторном запуске числа были другие) – функция srand(time(NULL)).

1. В функции `main()` объявите три переменные целого типа: два операнда (a, b) и результат умножения (c)::

```
#include <iostream>
#include <ctime>
```

```
using namespace std;

int main()
{
      srand(time(NULL));
      int a, b, c;
```

2. Далее объявите еще две переменных целого типа: для подсчета неправильных ответов (k = 0) и общего числа вопросов (n = 10):

```
      int k = 0, n = 10;
```

3. Добавьте реализацию цикла for, изучите содержимое тела цикла:

```
  for(int i = 1;i <= n ;i++)
  {
     // инициализация операндов случайными числами от 1 до 10[1]
         a = rand() % 10 + 1;
         b = rand() % 10 + 1;

         cout << a << " * " << b << " = ";
         cin >> c;

         if (a*b != c)
         {
               k++; // операция «инкремент», аналогично: k = k + 1
               cout << "Error! ";
               cout << a << " * " << b << " = " << a * b << endl;
         }
   }

  cout << "Count error: " << k << endl;
  return 0;

}
```

4. Запустите программу, проверьте работу алгоритма.

Упражнение 4. Расчет суммы чисел на заданном интервале

В этом упражнении требуется составить программу, реализующую следующее сумму:

$s = \sum_{i=1}^{100} i,$ для i, находящихся от 1 до k и от m до 100.

Для суммирования чисел в диапазоне воспользуйтесь циклом `for` и оператором перехода к следующей итерации `continue`:

```
      for (int i=1;i<=100;i++)
      {
            if ((i>k) && (i<m))
```

[1] формула для диапазона: r = min + rand() % (max – min +1)

```
            continue;
        s += i;
    }
```

Контрольные задания.

Задание 1. Определение суперпростого числа

Требуется реализовать алгоритм, проверяющий, является ли число суперпростым. Обоснуйте типы данных, используемые в алгоритме.

Задание 2. Проверка номера СНИЛС

Номер СНИЛС состоит из 9 цифр и двузначного контрольного числа. В основной части номера (первые 9 цифр) не может быть более чем двух одинаковых цифр подряд. Контрольное число СНИЛС рассчитывается следующим образом:

1. Каждая цифра СНИЛС умножается на номер своей позиции (позиция отсчитывается с конца), полученные произведения суммируются.
2. Если сумма меньше 100, то контрольное число равно самой сумме.
3. Если сумма равна 100 или 101, то контрольное число равно 00.
4. Если сумма больше 101, то вычисляется остаток от деления суммы на 101.
5. Если остаток меньше 100, то контрольное число равно остаток.
6. Если остаток равен 100, то контрольное число равно 00.

Требуется реализовать алгоритм, принимающую на вход строку символов и возвращающую True, если номер валидный, и False в противном случае.

Условие:

- На данном этапе реализовывать алгоритм в виде функции не требуется.
- Входная строка не может содержать произвольные символы, в том числе и пробелы с дефисами – реакция на подобный ввод является дополнительным заданием и в данный момент не требуется.

Задание 3. Размен монет

Известна монетная система России. Требуется выдать набор монет с наименьшим возможным количеством монет. Каждое число из набора можно использовать сколько угодно раз.

Для решения данной задачи рекомендуется использовать жадный алгоритм.

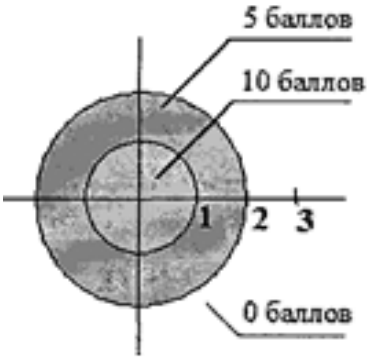
Задание 4. Стрельба по мишени

Требуется разработать программу, имитирующую стрельбу по мишени.

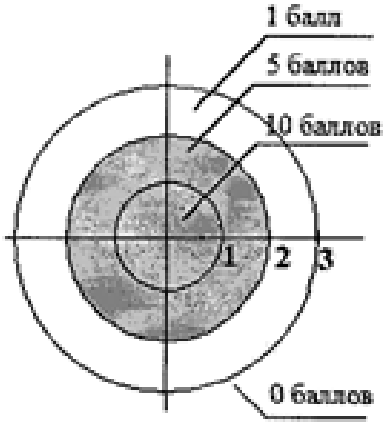
Пользователь вводит данные о выстреле в виде пары чисел – координат `x` и `y` заранее известное количество раз.

Повтор ввода следует организовать в цикле. После «стрельбы» пользователю выводится информация о сумме очков и его уровень как стрелка (например, снайпер, просто стрелок, новичок).

Вариант мишени выберите самостоятельно.



Вариант 1.



Вариант 2.

Дополнительные задания:

- реализовать центр мишени случайным значением, тогда стрелок не будет знать местонахождение мишени (стрельба «вслепую»);
- реализовать случайную помеху при выстреле, тогда стрелок будет использовать трудности при стрельбе.

Обязательные задания:

Внесите изменения в программу предыдущего задания согласно следующим требованиям:

- пользователь вводит данные о выстреле в виде пары чисел – координат `x` и `y` до тех пор, пока на наберет заранее установленное количество очков (например, 50),
- после окончания “стрельбы” вывести на экран число выполненных выстрелов и уровень стрелка (например, снайпер, просто стрелок, новичок).

Практическое занятие 3. Использование функций

На этом занятии изучается организация программ в соответствии с процедурным стилем программирования, правила описания, объявления и вызова функций.

Упражнение 1. Использование функции при организации программы

В этом упражнении вы создадите новую функцию для лучшей организации программы.

Задание 1. Реализация процедуры

1. Создайте новый проект и добавьте файл исходного кода.

2. В функции `main()` объявите строковую переменную и запросите имя с клавиатуры (это пример из занятия 1, обратите внимание на использование строкового типа):

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
 string name;
 cout << "What is your name?" << endl;
 cin >> name;
 cout << name << ", " << "hello!" << endl;
 return 0;
}
```

3. Объявите в начале программы до функции `main()` функцию, которая будет принимать один параметр строкового типа – имя и выводить строку приветствия на экран:

```
void privet(string name)
{
    cout << name << ", " << "hello!" << endl;
}
```

4. В функции вместо непосредственного вывода на экран результата приветствия вызовите новую функцию с передачей ей в качестве параметра введенное имя:

```
privet(name);
```

5. Постройте и протестируйте приложение.
6. Внесите изменения в программу: перенесите функцию `privet()` после функции `main()` и в начале программы перед функцией `main()` укажите прототип функции `privet()`:

```
void privet(string);
```

7. Постройте и протестируйте приложение.

Задание 2. Реализация возвращения значения функции

1. Создайте новый проект и скопируйте в его файл исходного кода проекта предыдущего задания.
2. Объявите в начале программы до функции `main()` функцию, которая будет принимать один параметр строкового типа – имя и возвращать измененную строку приветствия на экран:

```
string privet(string name)
{
```

```
    string str = name + ", " + "hello!\n";
    return str;
}
```

3. В функции `main()` вызовите функцию с передачей ей в качестве параметра введенное имя и присвойте возвращаемый результат новой переменной:

```
string nameOut = privet(name);
```

4. В функции `main()` реализуйте с помощью стандартного объекта `cout` вывод переменной `nameOut` на экран.
5. Постройте и протестируйте приложение.
6. Сравните реализацию функций в первом и втором заданиях.

Упражнение 2. Перегрузка функций

В этом упражнении вы используете механизм перегрузки функций. Для перегрузки функций требуется определить функции с одним и тем же именем, которые отличаются количеством параметров и/или их типом.

1. В проекте упражнения 1 добавьте (не замените имеющуюся, а именно добавьте новую) вторую функцию `privet()`, но с двумя параметрами:

```
void privet(string name, int k)
{
  cout << name << ", " << "hello! " << "you input " << k << endl;
}
```

2. В функции `main()` объявите целочисленную переменную и реализуйте ввод ее значения:

```
int main ()
{
  ...
  int k;
  ...
 cout << "Input number:" << endl;
 cin >> k;
```

3. Добавьте вызов перегруженной функции:

```
privet(name);
privet(name,k);
```

4. Постройте и протестируйте приложение. Обратите внимание на работу перегруженных функций.

Упражнение 3. Реализация сложных алгоритмов с помощью функций

В этом упражнении вы реализуете в виде функции алгоритм бинарного поиска для вычисления корня указанной степени.

➢ Бинарный поиск производится в упорядоченной последовательности. При таком поиске искомый ключ сравнивается с ключом среднего элемента в массиве и если они

равны, то поиск успешен. В противном случае поиск осуществляется аналогично в левой или правой частях последовательности. Алгоритм может быть определен в рекурсивной и не рекурсивной формах. Бинарный поиск также называют поиском методом деления отрезка пополам или дихотомии.

Дано действительное число `a` и натуральное `n`.

Требуется вычислить корень n-й степени из числа `a`, используя вещественный бинарный поиск.

Формат входных данных:

С клавиатуры через пробел вводится два числа:

1. a – действительное, больше или равное 1, не превосходит 1000, задано с точностью до 6 знаков после десятичной точки;

2. n – натуральное, не превосходящее 10.

Формат выходных данных:

Число с точностью не менее 6 знаков после запятой.

1. Создайте новый проект и объявите функцию, реализующую алгоритм бинарного поиска для нахождения значения корня степени n:

```
long double firBinSearch(double a, int n)
{
    double L = 0;
    double R = a;
    while (R - L > 1e-10)
    {
        double M = (L + R) / 2;
        if (pow(M, n) < a)
        {
            L = M;
        }
        else
        {
            R = M;
        }
    }
    return R;
}
```

2. В функции `main()` реализуйте ввод исходных данных, вызов функции и вывод на экран результата в требуемом формате.

Упражнение 4. Применение рекурсивной функции

➢ Рекурсия — это такой способ организации обработки данных, при котором функция вызывает сама себя непосредственно, либо с помощью других функций. Рекурсивный алгоритм разбивает задачу на части, которые по своей структуре являются такими же как исходная задача, но более простыми. Рекурсивные функции должны завершаться при достижении некоторого условия.

➢ Любой алгоритм, реализованный в рекурсивной форме, может быть переписан в итерационном виде и наоборот (останется вопрос, надо ли это, и насколько это будет это эффективно).

В этом упражнении вы реализуете рекурсивно алгоритмы для известных задач, ранее решенных вами итерационно – вычисление суммы чисел от 1 до любого целого положительного числа (Занятие 2, упражнение 3) и определение наибольшего общего делителя по алгоритму Евклида (Занятие 2, упражнение 2).

Задание 1. Сумма чисел

1. Создайте новый проект и объявите функцию, реализующую алгоритм вычисления суммы чисел от 1 до любого целого положительного числа n:

```
int addNumders(int n)
{
     if (n == 1) return 1;        // выход из рекурсии
     else return (n + addNumders(n - 1));
}
```

2. В функции `main()` реализуйте ввод требуемого числа, вызов функции и вывод на экран результата.
3. Замените операцию сложения на вычитание. Запустите программу и объясните полученный результат. Верните обратно оператор сложения.
4. Объявите вторую функцию с тем же именем (она будет перегруженной), которая должна будет суммировать числа на интервале от первого числа до второго.
5. В функции `main()` реализуйте ввод двух переменных – интервала для которого вычисляется сумма чисел, числа, вызов функции и вывод на экран результата.

Задание 2. Алгоритм Евклида

1. В этом же проекте объявите функцию, реализующую алгоритм Евклида: определение наибольшего общего делителя с использованием операции вычисления остатка от деления:

```
int gcd(int m, int n)
{
      if (n == 0) return m;
      return gcd(n, m % n);
}
```

2. В функции `main()` реализуйте ввод двух чисел, вызов функции и вывод на экран результата.

Контрольные задания.

Задание 1. Проверка номера СНИЛС

Требуется разработать функцию проверки номера СНИЛС по ранее реализованному алгоритму (см. Практическое занятие 2, задание 2), принимающую

на вход строку символов и возвращающую True, если номер валидный, и False в противном случае.

Обязательные условия:

- Входная строка может содержать произвольные символы, причем, если в строке встречаются пробелы и дефисы, функция должна игнорировать их и проводить проверку по заданию.
- Если же в строке встречаются какие-либо символы кроме цифр, пробелов и дефисов, функция должна возвратить False.

Задание 2. Применение итерации реализации функции

Требуется реализовать функцию вычисления кубического корня в двух вариантах: в первом, используя стандартную функцию `pow(a,1.0/3)`, а во втором – итерационную формулу:

$$x_i = \frac{1}{3}\left(\frac{a}{x_{i-1} * x_{i-1}} + 2 * x_{i-1}\right).$$

В функции `main()` протестируйте работу созданных функций.

Задание 3. Работа с различными треугольниками

Требуется написать программу, рассчитывающее площадь треугольников двух типов: равностороннего и разностороннего.

Для решения задачи реализуйте перегруженные функции: первая должна принимать один параметр – сторону и вычислять площадь равностороннего треугольника, вторая – принимать три параметра-стороны треугольника для вычисления площади разностороннего треугольника.

Выбор типа треугольника реализуйте с помощью диалога с пользователем.

Задание 4. Рекурсивная функция суммы ряда

Разработайте рекурсивную функцию вычисления суммы ряда

$S = 5 + 10 + 15 + \ldots + 5 \cdot n$,

при $n>0$.

Задание 5. Применение рекурсии для перевода целого числа в двоичный код

Требуется написать рекурсивную функцию перевода целого положительного числа из десятичной системы исчисления в двоичную.

Классический алгоритм перевода: нужно входной параметр `num` делить на 2 до тех пор, пока `num>0`. При каждом делении нужно выделять остаток с помощью выражения num%2.

Практическое занятие 4. Использование указателей и ссылок

Упражнение 1. Передача параметров

В этом упражнении исследуется механизм передачи параметров в функцию так, чтобы изменения параметров внутри этой функции привели бы к изменению исходных параметров. Сделать это можно, передав в функцию адрес переменной (указатель) или ссылку на нее. В этих случаях передается копия значения адреса на область памяти. На один и тот же участок памяти может существовать множество ссылок, и с помощью каждой из них можно поменять находящееся там значение.

Требуется реализовать возможность одновременно изменить значения двух переменных – координат `x` и `y` некоторой точки на плоскости.

1. Создайте новый проект, добавьте пустой файл исходного кода.
2. До функции `main()` определите функции, первая из них принимает параметры по умолчанию, вторая принимает указатель, третья – ссылку, первый параметр, передаваемый в каждую функцию – значение, на которое будут увеличены значения `x` и `y`:

```
void fum_value(double k, double x, double y)
{
   x = x + k;
   y = y + k;
}
void fum_ptr(double k, double *x, double *y)
{
  *x = *x + k;
  *y = *y + k;
}
void fum_ref(double k, double &x, double &y)
{
   x = x + k;
   y = y + k;
 }
```

3. Определите функцию, которая будет выводить значения параметров:

```
void print(double x, double y)
{
     cout << "x = " << x << "; y = " << y << endl;
}
```

4. В функции `main()` объявите, проинициализируйте переменные и вызовите поочередно созданные ранее функции с передачей параметров:

```
int main()
{
  double k = 2.5;
  double xv = 10;
```

```
  double yv = 10;

  print(xv, yv);

  fum_value(k, xv, yv); // Передача в функцию обычного параметра
  print(xv, yv);

  fum_ptr(k, &xv, &yv); // Передача в функцию параметра указателя
  print(xv, yv);

  fum_ref(k, xv, yv); // Передача в функцию параметра ссылки
  print(xv, yv);

  return 0;
}
```

5. Постройте и протестируйте приложение. Обратите внимание на результат работы функций.

Упражнение 2. Реализация функции обмена значений

В этом упражнении исследуется механизм обмена значений двух переменных с помощью функции так, чтобы изменения параметров внутри этой функции привели бы к изменению (обмена) исходных параметров.

Требуется реализовать возможность обмена двух переменных с помощью функции.

1. Создайте новый проект, добавьте пустой файл исходного кода.
2. До функции `main()` объявите функции (прототипы), первая принимает два указателя на целый тип, вторая принимает две ссылки:

```
void swap(int*, int*);
void swap(int&, int&);
```

3. После функции определите требуемые функции:

```
void swap (int *x, int *y)
{
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
void swap (int &x, int &y)
{
   int temp;
   temp = x;
   x = y;
```

```
   y = temp;
}
```

4. В функции `main()` объявите две переменные и вызовите реализованные функции, передав в первом случае адреса (принимает функция с указателями), а во втором – параметры (принимает функция со ссылками):

```
int x = 5, y = 10;
swap(&x,&y);
swap(x,y);
```

5. Добавьте код вывода значений переменных на экран после каждого вызова.
6. Постройте и запустите приложение. Обратите внимание на результат работы функций.

Контрольные задания

Задание 1. Вычисление корней квадратного уравнения

Требуется реализовать функцию вычисления действительных корней квадратного уравнения.

- Функция должна возвращать значение 1, если корни найдены, значение нуля, если оба корня совпадают, и значение -1, если корней не существует.
- Значения корней уравнений должны возвращаться в качестве аргументов функции, передаваемых по ссылке.

Прототип функции должен выглядеть следующим образом:

```
int Myroot(double, double, double, double&, double&);
```

Для тестирования работы данной функции в функции `main()` реализуйте ее вызов с передачей параметров согласно определения функции. Возвращаемый результат (целочисленный флаг) используйте при организации вывода сообщений о полученных корнях уравнения, например, «Корней уравнения нет», «Корень уравнения один x1 = x2 = результат», «Корни уравнения x1 = результат, x2 = результат».

Задание 2. Реализация ввода данных

Требуется реализовать функцию `Input(a,b)` так, чтобы она была использована в программе следующим образом:

```
int main()
{
    int a, b;
    if(Input(a,b)== false) // if(!Input(a,b))
    {
     cerr << "error";
     return 1;
    }
    int s = a + b;
```

```
    return 0;
}
```

В функции `Input(a,b)` должен быть реализован ввод данных с клавиатуры с проверкой корректности ввода (правила проверки корректности кода установите на ваше усмотрение).

Практическое занятие 5. Работа с массивами

Упражнение 1. Обработка данных массива

В этом упражнении вы создадите массив, заполните его числами и выполните обработку данных.

1. Создайте новый проект, добавьте пустой файл исходного кода.
2. В функции `main()` объявите константу, равную 10, она будет задавать размер массива.

```
const int n = 10;
```

3. Объявите массив целых чисел размером n:

```
int mas[n];
```

4. С помощью цикла `for` реализуйте заполнение массива с клавиатуры:

```
for (int i=0; i<n; i++)
{
      cout << "mas[" << i << "]=";
      cin >> mas[i];
}
```

5. Определите сумму всех элементов массива:

```
int s = 0;
for (int i=0; i<n; i++)
{
      s += mas[i];
}
```

6. Выведите значение суммы на экран.
7. Постройте и протестируйте приложение.
8. Добавьте новую переменную и присвойте среднее значение элементов массива, выведите ее на экран
9. Добавьте в функции `main()` новые возможности обработки данных массива:
    a. расчет суммы отрицательных элементов,
    b. расчет суммы положительных элементов,
    c. расчет суммы элементов с нечетными индексами,
    d. расчет суммы элементов с четными индексами.
10. Дополнительные задания:

a. найти максимальный и минимальный элементы массива и вывести их индексы,
b. рассчитать произведение элементов массива, расположенных между максимальным и минимальным элементами

11. Постройте и протестируйте приложение.

Упражнение 2. Сортировка массива

В этом упражнении вы реализуете простой алгоритм сортировки элементов массива – сортировка выбором.

➢ Суть сортировки выбором состоит в поиске минимального (при упорядочивании по возрастанию) значения элемента в массиве, и перемещения этого значения в начало массива (начало массива это позиция куда перемещается найденное минимальное значение и “начало” в алгоритме с каждым шагом цикла будет смещаться в сторону хвоста массива). На первой итерации цикла, найденное минимальное значение меняется местами со значением в нулевой ячейке массива. На второй итерации “начало” уже будет указывать на следующую (первую) ячейку и так далее.

1. Создайте новый проект, добавьте пустой файл исходного кода.
2. В функции `main()` объявите константу, равную 10, она будет задавать размер массива.

```
const int N = 10;
```

3. Объявите массив целых чисел размером N и проинициализируйте его произвольными значениями:

```
int a[N]={ 1, 25, 6, 32, 43, 5, 96, 23, 4, 55 };
```

4. Объявите две переменные для записи минимального значения и для обмена значениями:

```
int min = 0; // для записи минимального значения
int buf = 0; // для обмена значениями
```

5. Реализуйте с помощью цикла for перебор всех элементов массива, а в нем создайте второй цикл for для реализации перестановок элементов (обратите внимание на применение операции выбора ?):

```
for (int i = 0; i < N; i++)
{
    min = i; // номер текущей ячейки, как ячейки с минимальным значением
// в цикле найдем реальный номер ячейки с минимальным значением
     for (int j = i + 1; j < N; j++)
         min = ( a[j] < a[min] ) ? j : min;
// перестановка этого элемента, поменяв его местами с текущим
     if (i != min)
     {
      buf = a[i];
      a[i] = a[min];
      a[min] = buf;
```

```
        }
    }
```

6. Реализуйте вывод отсортированного массива на экран. Для этого вместо стандартного варианта цикла for можно применить цикл с перебором массива (foreach):

```
for (int i: a)
    cout << i << '\t';
```

7. Постройте и протестируйте приложение.

Упражнение 3. Использование указателя на функцию

В этом упражнении вы создадите массив, который в зависимости от выбора пользователя сортируется по убыванию или возрастанию. Выбор будет реализован с помощью передачи указателя на функцию. Для сортировки массив будет передаваться в функцию в качестве параметра.

1. Создайте новый проект, добавьте пустой файл исходного кода.
2. В функции `main()` объявите константу, равную 10, она будет задавать размер массива.
3. Далее объявите переменную с помощью, которой будет выполняться выбор направления сортировки. Начальное значение переменной установите равным нулю.
4. Создайте массив и проинициализируйте его произвольными значениями.
5. Реализуйте возможность диалога с пользователем, код этой части программы может быть следующим:

```
#include <iostream>
using namespace std;

int main()
{
    const int N = 10;
    int my_choose = 0;
    int A[N] = {9,8,7,6,1,2,3,5,4,9};
      cou t<< "1. Сортировать по возрастанию\n";
      cout << "2. Сортировать по убыванию\n";
      cin >> my_choose;
      cout << "Исходные данные: ";
return 0;
}
```

6. Операцию отображения массива на экране реализуйте отдельной функцией, первый параметр – массив, второй – размер массива, причем оба параметра сделайте константными:

```
void show_array(const int Arr[],const int N)
{
```

```
    for (int i = 0; i < N; i++)
    cout << Arr[i] <<"  ";
    cout << "\n";
}
```

7. В начале программы добавьте прототип функции.
8. В конце функции `main()` добавьте вызов функции отображения массива, передав ей массив и его размер:

```
show_array(A,N);
```

9. Постройте и запустите приложение. Проверьте работу функции отображения исходного массива.
10. Добавьте две функции, определяющие направление сравнения (на них будет использован указатель). Не забудьте добавить прототипы функций.

```
bool from_min(const int a, const int b)
{
    return a>b;
}
bool from_max(const int a,cons tint b)
{
    return a<b;
}
```

11. Добавьте функцию обменной (пузырьковой) сортировки. Первые два параметра, передаваемые в функцию – константный массив и его размер. Третий параметр булевого типа должен принимать указатель на функцию с двумя параметрами, причем сам указатель заключен в круглые скобки, а параметры в отдельные круглые скобки, вплотную к этому указателю.

```
void bubble_sort(int Arr[],const int N, bool (*compare)(int a,int b))
{
    for (int i=1;i<N;i++)
    {
        for (int j=0;j<N-1;j++)
        {
            if ((*compare)(Arr[j],Arr[j+1])) swap(Arr[j],Arr[j+1]);
        }
    }
}
```

12. Изучите код, реализующий алгоритм обменной сортировки, инструкцию

```
if ((*compare)(Arr[j],Arr[j+1]))
```

можно прочитать так: если функция возвращает 1, то выполнить функцию `swap`. Получается, что в функцию автоматически передаются два подряд идущих значения, они сравниваются, и возвращается результат сравнения – истина или ложь.

13. В функции `main()` после вызова функции отображения массива добавьте конструкцию `switch` для определения поведения программы в зависимости от выбора пользователя и реализуйте вызов функции сортировки, передав ей в третий параметр в первом случае указатель на функцию `from_min`, во втором – на функцию `from_max`

```
switch (my_choose)
{
   case 1: bubble_sort(A,N,from_min); break;
   case 2: bubble_sort(A,N,from_max); break;
   default: cout<<"\rНеизвестная операция ";
}
```

14. Повторно вызовите функцию отображения массива

```
show_array(A,N);
```

15. Постройте и запустите приложение. Проверьте работу программы в зависимости от выбора пользователя.

Использование массива указателей на функции

В этой части упражнения Вы с помощью массива указателей на функции замените конструкцию `switch`. Выбор требуемой функции, определяющей направление сортировки, будет реализован обращением к элементу массива.

1. В начале функции `main()` объявите массив из двух указателей на функции, где функции принимают два целых аргумента, а тип возвращаемого параметра – `bool`:

```
bool (*from_f[2])(int, int) = { from_min,from_max };
```

2. Вместо конструкции `switch`вызовите функцию сортировки и третьим параметром передайте разыменованный указатель, расположенный в элементе массива с индексом `my_choose - 1`:

```
bubble_sort(A,N,(*from_f[my_choose - 1]));
```

3. Постройте и запустите приложение. Проверьте работу программы в зависимости от выбора пользователя.
4. Допускается не разыменовывать указатель на функцию для вызова функции, на которую он ссылается, потому что компилятор это сделает сам. Удалите символ * при передаче параметра в функцию и повторно протестируйте программу.

Упражнение 4. Реализация динамического массива

Требуется в программах предыдущих упражнений вместо заранее инициализированного массива реализовать возможность пользователю ввести необходимый размер массива, а затем заполнить его.

Указание. Для создания динамического массива можно использовать следующий код, например:

```
int* myArray = new int[n];
```

В данном примере `myArray` является указателем на массив из `n` элементов.

Не забудьте после использования массива освободить динамическую память.

Упражнение 5. Применение структур данных для хранения элементов

В этом упражнении вы выберите подходящий тип для хранения результатов (в виде структур данных на примере массива и вектора) тестирования таблицы умножения (занятие 2, задание 3).

Задание 1. Сохранение результатов в массиве

В программе тестирования знаний таблицы умножения требуется сохранять введенные ответы и затем выводить результаты на экран.

1. Подумайте, в какой структуре данных можно сохранять вводимые ответы?

Ход рассуждений:

- Данные одного типа?
- Да, планируется сохранять значения переменой *c*, а она целого типа.
- Заранее известно сколько должно храниться значений?
- Да, по замыслу теста предполагается 10 вопросов.
- Есть ли особые требования к размещению данных в памяти?
- Нет, рекомендуется использовать наиболее удобный и быстрый способ хранения и обработки данных.

Итог рассуждений: наиболее подходящая структура данных – массив.

2. В функции main() программы тестирования знаний таблицы умножения (занятие 2, задание 3) замените объявление переменной *n* на объявление константы и далее объявите массив для хранения десяти целых чисел:

```
int main()
{
      srand(time(NULL));
      int a, b, c;
      int k = 0;

      const int n = 10;
      int mas[n];
```

3. В заголовок цикла for внесите изменения: вместо `int i=1; i<=n` укажите индексацию с нуля (так как массивы индексируются с нулевого индекса) и условие цикла не включает n: `int i=0; i<n`

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

4. Далее в теле цикла после ввода переменной *c* (ответа на тест) добавьте выражение, присваивающее текущему элементу массива значение этой переменной c:

```
cin >> c;
```

```
mas[i] = c;
```

5. В функции main() перед выводом количества ошибок создайте новый цикл for для вывода на экран всех введенных ответов:

```
cout << "\nAll: ";
for (int i=0; i<n; i++)
{
      cout << mas[i] << ends;
}
```

6. Запустите программу, проверьте ее работу, должны быть выведены элементы массива – введенные вами ответы.

Задание 2. Сохранение набора результатов неизвестного размера

Теперь в программе тестирования знаний таблицы умножения требуется также сохранять введенные правильные и не правильные ответы и затем выводить результаты на экран.

1. Подумайте, в какой структуре данных можно сохранять правильные и не правильные ответы?

Ход рассуждений:

- Данные одного типа?
- Да, планируется сохранять значения переменой *c*, а она целого типа.
- Заранее известно сколько должно храниться значений?
- Нет, по замыслу теста предполагается 10 вопросов, сколько из них правильных и не правильных не известно.
- Есть ли особые требования к размещению данных в памяти?
- Нет, рекомендуется использовать наиболее удобный и быстрый способ хранения и обработки данных.

Итог рассуждений: наиболее подходящая структура данных – **вектор** – динамическая структура, позволяющая доступ к элементу по индексу.

2. В начале программы подключите заголовочный файл, в котором описан вектор:

```
#include <vector>
```

3. В функции main() после объявления массива объявите два вектора (они изначально пустые) для хранения целых чисел:

```
vector<int> v1;
vector<int> v2;
```

4. В цикле for в конструкции if укажите код для добавления неправильного введенного ответа в вектор v2:

```
if (a*b != c)
{
      v2.push_back(c);
      k++;
```

5. После блока if добавьте блок else для реализации ветви в случае введения правильного ответа и в этом блоке укажите код для добавления правильного введенного ответа в вектор v1:

```
else
{
      v1.push_back(c);
}
```

6. После цикла выводящего все значения массива добавьте два цикла for для аналогичного вывода правильных и неправильных ответов (для определения условия продолжения цикла примените для вектора стандартную функцию size()):

```
cout << "\nGood: ";
for (int i = 0; i < v1.size(); i++)
{
      cout << v1[i] << ends;
}

cout << "\nBad: ";
for (int i = 0; i < v2.size(); i++)
{
      cout << v2[i] << ends;
}
```

7. Запустите программу, проверьте ее работу, должны быть выведены элементы массива – введенные вами ответы и элементы векторов – правильные и неправильные ответы.

Контрольные задания

Задание 1. Передача массива в функцию

Реализуйте созданный вами функционал в упражнениях 1 и 2 в виде соответствующих функций, которые будут принимать массив и возвращать требуемые результаты.

Функция должна принимать два параметра: целочисленное значение – размер массива и сам массив.

Задание 2. Возврат массива из функции

➢ В качестве результата работы функции может быть указатель на массив, создаваемый в этой функции.

Дано: в функции `main()` объявлены два массива равного размера.

Требуется: изучите содержимое функции `main()` и реализуйте функцию `max_vect()` согласно сценария функции `main()`, так чтобы она возвращала адрес на созданный ею массив с элементами – большими значениями на соответствующих индексах исходных массивов, например, для заданных массивов результирующий массив должен быть равен 7, 6, 5, 4, 5, 6, 7, 3.

Таким образом, в результате вызова функции `max_vect(kc,a,b)` возвращается массив, состоящий из больших элементов, стоящих на соответствующих местах в исходных массивах.

```
int main()
{
 int a[]={1,2,3,4,5,6,7,2};
 int b[]={7,6,5,4,3,2,1,3};

 int kc = sizeof(a)/sizeof(a[0]); //Количество элементов массива
 int *c;                          //Указатель на результирующий массив
 c = max_vect(kc,a,b);            //вызов функции для создания массива

 for (int i = 0;i < kc; i++)              //Вывод результата.
      cout << c[i] << " ";
 cout << endl;
 delete []c;                        //Возврат памяти.
}
```

Задание 3. Реализация алгоритмов поиска методом транспозиции

Последовательный поиск предполагает последовательный просмотр всех записей множества, организованного как массив. Улучшением рассмотренного метода является метод транспозиции: каждый запрос к записи сопровождается сменой мест этой и предшествующий записи. В итоге наиболее часто используемые записи постепенно перемещаются в начало массива и при последующем обращении к ним они находятся значительно быстрее.

Реализуйте функцию поиска в массиве заданной размерности конкретного элемента (key) с использованием алгоритма транспозиции.

Задание 4. Передача параметров в программу

Требуется написать программу, которая, принимая в качестве параметров два целочисленного операнда, возвращала бы их сумму или произведение в зависимости от дополнительного параметра-флага, например, вызов

```
nameProg –a 12 45
```

возвращал бы результат суммирования, а вызов

```
nameProg –m 12 45
```

возвращал бы результат произведения.

Указание: требуется реализовать проверки вводимых данных:

- по количеству параметров – если количество переданных функции аргументов `argc` меньше четырёх (в качестве первого аргумента выступает имя программы),
- по формату – если второй элемент массива параметров `argv[]` не равен строке `–a` или `–m`.

Рекомендация: сравнение удобно проводить при помощи функции `strncmp()` из библиотеки `<cstring>`, которая имеет следующий синтаксис:

```
int strncmp(const char *str1, const char str2, size_t count)
```

Функция производит лексикографическое сравнение первых `count` символов строк `str1` и `str2` и возвращает 0, если строки равны, значение меньше нуля, если `str1` меньше `str2`, и значение больше нуля — в противном случае.

Для конвертации строки в целое число можно применить функцию `atoi`:

```
int atoi(const char* str)
```

Передаваемая строка должна содержать корректную запись целого числа, в противном случае возвращается нуль.

Практическое занятие 6. Работа с файлами

Упражнение 1. Запись и чтение данных из бинарного файла

В этом упражнении вы запишите данные измерений в бинарный файл, а затем реализуете считывание данных из этого файла и подсчитаете их сумму.

1. Создайте новый проект и добавьте файл исходного кода.
2. В методе `main()` объявите переменную – сумму чисел, константу – размер массива и сам массив:

```
double sum = 0;
int const n = 100;
double nums[n];
```

3. В цикле заполните массив случайными числами:

```
for (int i = 0; i < n; i++)
 {
   nums[i] = (rand() % 100);
 }
```

4. Создайте объект **ofstream** и свяжите его с определенным файлом на диске (использование объектов **ofstream** требует включения в программу файла заголовка **fstream**):

```
ofstream out ("test", ios::out | ios::binary);
if (!out) {
      cout << "Файл открыть невозможно\n";
      return 1;
}
```

Значение режима открытия файла `ios::binary` предотвращает преобразование символов.

5. Вызовите функцию **write()**, которая записывает в поток из буфера, который определен указателем на буфер – `(char*)nums`,заданное число байтов `sizeof(nums))`:

```
out.write((char *) nums, sizeof(nums));
```

Приведение типа к `(char*)` при вызове функции **write**() необходимо, если буфер вывода не определен как символьный массив.

6. Закройте поток с помощью функции `close()`:

```
out.close();
```

7. Для открытия файла используйте объект **ifstream**:

```
ifstream in("test", ios::in | ios::binary);
  if(!in) {
  cout << "Файл открыть невозможно";
  return 1;
  }
```

8. С помощью функции **read**() считайте из вызывающего потока столько байтов, сколько задано в `sizeof(nums)` и передайте их в буфер, определенный указателем буфер `(char *) &nums`:

```
in.read((char *) &nums, sizeof(nums));
```

9. Реализуйте обработку полученных данных – подсчитайте их сумму и выведите результаты на экран:

```
int k = sizeof(nums)/sizeof(double);
      for (int i=0; i<k; i++)
      {
            sum = sum + nums[i];
            cout << nums[i] <<' ';
      }
cout << "\nsum = " << sum << endl;
```

10. Закройте поток:

```
in.close();
```

11. Постройте и протестируйте приложение.

Контрольные задания

Задание 1. Запись текста в файл

Требуется написать программу для записи небольшого стихотворения с клавиатуры в текстовый файл.

Задание 2. Сохранение данных в текстовый файл

В решении упражнения 2 занятия 5 создайте текстовый файл и запишите в него два массива: исходный и отсортированный.

Практическое занятие 7. Применение структур и кортежей

Упражнение 1. Реализация структуры Distance

В этом упражнении вы создадите структуру `Distance`, определяющую длину в английской системе мер. В английской системе мер основными единицами измерения длины служат фут и дюйм, причем один фут равен 12 дюймам. Расстояние, равное, например, 15 футам и 8 дюймам, на экран выведете как 15 '- 8". Дефис в данной записи будет служить для визуального разделения значений футов и дюймов.

Для тестирования структуры `Distance` в методе `Main` класса `Program`:

- определите три переменные типа `Distance` и две из них инициализируйте с помощью значений, вводимых с клавиатуры.
- присвойте третьей переменной значение суммы первых двух переменных (при реализации операции сложения пока не учитывайте, что один фут равен 12 дюймам, это будет сделано в следующей работе) и выведите результат на экран.

1. Создайте новое приложение и добавьте файл исходного кода.
2. До функции `main()` объявите структуру `Distance`, представляющую расстояние в английской системе мер, поля структуры – футы и дюймы:

```
#include <iostream>
#include <windows.h>
using namespace std;


struct Distance
{
   int feet;
   double inches;
};
```

3. После объявления структуры добавьте функцию сложения двух переменных типа `Distance`, при этом реализуйте возможность увеличения числа футов при переполнении дюймов:

```
Distance AddDist(Distance d1, Distance d2)
{
     Distance d;
     d.feet = d1.feet + d2.feet;
     d.inches = d1.inches + d2.inches;
     if (d.inches>= 12.0 )
     {
           d.inches -= 12.0;
           d.feet++;
     }
```

```
    return d;
 }
```

4. Добавьте функцию для ввода значений футов и дюймов:

```
Distance InputDist()
{
    Distance d;
    cout << "\nВведите число футов: ";
    cin >> d.feet;
    cout << "Введите число дюймов: ";
    cin >> d.inches;
    return d;
}
```

5. Добавьте еще одну функцию – вывод информации о переменной структуры:

```
void ShowDist(Distance d)
{
  cout << d.feet << "\'-" << d.inches << "\"\n";
}
```

6. В функции `main()` объявите переменную типа `Distance` и для ее инициализации вызовите требуемую функцию, затем объявите вторую переменную с одновременной инициализацией:

```
int main()
{
   Distance d1 = InputDist();
   Distance d2 = { 1, 6.25 };
 ...
   return 0;
}
```

7. Объявите третью переменную типа `Distance` и присвойте ей результат сложения двух первых переменных, используя функцию `AddDist`:

```
Distanced3 = AddDist(d1,d2);
```

8. Для вывода значений трех переменных вызовите последовательно функцию `ShowDist`:

```
ShowDist(d1);
ShowDist(d2);
ShowDist(d3);
```

9. Постройте и запустите приложение. Проверьте работу функций и правильность выполнения операции сложения.
10. Перенесите определение функции `ShowDist()` в структуру:

```
void ShowDist()
{
```

```
cout << feet << "\'-" << inches << "\"\n";
}
```

и вызовите ее для структурных переменных `d1`, `d2` и `d3`, например:

```
d1.ShowDist();
```

11. Постройте и протестируйте приложение.

Упражнение 2. Передача структуры в функцию по ссылке

В этом упражнении вы измените тип передачи структурной переменной в функцию. В первом упражнении использовалась передача по значению, в результате создавалась копия, которая участвовала в реализации алгоритма внутри функции. В этом случае на создание копии тратится память и время. В этом упражнении будет реализована передача объекта типа структура в функцию по ссылке.

1. Укажите в заголовке метода сложения константные ссылки, которые запрещают изменение объекта внутри функции:

```
Distance AddDist(const Distance &d1, const Distance &d2)
{...}
```

2. В этом случае копия передаваемого объекта не создается, но изменить объект внутри функции невозможно.
3. Постройте и запустите приложение.

Упражнение 3. Использование массива структур

В этом упражнении вы создадите массив переменных типа `Distance`.

1. В конце функции `main()` объявите переменную – размер массива и запросите ее значение с клавиатуры:

```
int n;
cout << "Введите размер массива расстояний " ;
cin >> n;
```

2. Объявите динамический массив переменных типа `Distance`:

```
Distance * masDist = new Distance[n];
```

3. В цикле реализуйте вызов функции ввода значений футов и дюймов для каждого элемента массива:

```
for (int i = 0; i<n; i++)
{
    masDist[i] = InputDist();
}
```

4. Также в цикле реализуйте вызов функции вывода значений на экран для каждого элемента массива

```
for (int i = 0; i<n; i++)
{
    ShowDist(masDist[i]);
```

```
}
```

5. Укажите оператор высвобождения памяти, занимаемой массивом:

```
delete[] masDist;
```

6. Постройте и запустите приложение. Протестируйте работу массива.
7. Реализуйте в программе подсчет и вывод на экран суммы всех расстояний, входящих в массив.

Упражнение 4. Применение кортежей для представления данных

В этом упражнении вы реализуете функцию, которая печатает кортеж (коллекцию разнородных типов фиксированного размера) из трех значений. Как правило, кортежи используются в математике для обозначения конечной последовательности элементов, а в языках программирования они применяются для связывания элементов различных типов.

1. Создайте новое приложение с файлом исходного кода.
2. В функции `main()` создайте три вектора – они будут представлять исходные данные для формирования кортежей.

```
#include <iostream>
#include <windows.h>
#include <string>
#include <tuple>
#include <vector>

using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);

    vector<string> v1{ "one", "two", "three", "four", "five", "six" };
    vector<int> v2 = { 1, 2, 3, 4, 5, 6 };
    vector<double> v3 = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
    return 0;
}
```

3. В функции `main()` создайте переменную кортежа типа `std::tuple` – объявление объекта обычно указывает типы, из которых будет состоять кортеж, и порядок, в котором к ним будет осуществляться доступ. `std::tuple` можно инициализировать разными конструкторами, в этом упражнении используется функция `std::make_tuple`, которая возвращает объект `std::tuple`, содержащий заданные значения:

```
auto t0 = make_tuple(v1[0], v2[0], v3[0]);
```

4. Объявите функцию с именем `printTupleOfThree()`, которая печатает кортеж, для доступа к значениям кортежа используется функция

get<индекс>(кортеж), причем значение индекса должно быть целочисленным константным выражением:

```
void printTupleOfThree(tuple<string, int, double> t)
{
    cout << "("
        << std::get<0>(t) << ", "
        << std::get<1>(t) << ", "
        << std::get<2>(t) << ")" << endl;
}
```

5. В функции `main()` вызовите функцию печати кортежа, передав в его качестве параметра:

```
printTupleOfThree(t0);
```

6. Постройте и протестируйте приложение.

Упражнение 5. Возвращение кортежа из функции

В этом упражнении вы создадите функцию – свою версию формирования кортежа на основе трех переменных различного типа.

Для удобства работы с кортежами можно использовать ключевое слово typedef, которое позволяет создать псевдоним для любого типа данных и использовать его вместо фактического имени типа.

1. В программу предыдущего упражнения добавьте следующее объявление:

```
typedef tuple<string, int, double> Tuple;
```

2. Теперь можно сократить запись при объявлении методов, использующих данный тип кортежа, исправьте объявление метода `printTupleOfThree()`:

```
void printTupleOfThree(Tuple t)
```

3. Проверьте, что программа работает без изменений.

Замечание. Для объявления псевдонима в C++11 введен новый улучшенный синтаксис, который имитирует способ объявления переменных – *type alias:*

using identifier = type;

4. Замените созданный typedef на новый вариант:

```
using Tuple = tuple<string, int, double>;
```

5. Проверьте, что программа работает без изменений.
6. Добавьте объявление следующей функции (логика формирования кортежа может быть и другой):

```
Tuple funtup(string s, int a, double d)
{
    s.append("!");
    return make_tuple(s, a, d*10);
}
```

7. В функции `main()` вызовите новую функцию для создания нового кортежа на основе исходных данных :

```
auto t1 = funtup(v1[1], v2[1], v3[1]);
```

8. И выведите на экран новый кортеж:

```
printTupleOfThree(t1);
```

9. Постройте и протестируйте приложение.
10. Добавьте еще одну функцию, которая будет принимать кортеж, изменять содержимое его переменных (на ваше усмотрение) и возвращать новый кортеж того же типа.
11. Добавьте еще одну функцию, которая будет принимать кортеж, изменять содержимое его переменных, формировать две переменные (на ваше усмотрение) и возвращать новый кортеж другого типа, состоящий из этих двух элементов.

Контрольные задания

Задание 1. Структура Time

Создайте структуру с именем Time. Три ее поля, имеющие тип int, назовите hours, minutes и seconds. Напишите программу, которая просит пользователя ввести время в формате часы, минуты, секунды. Можно запрашивать на ввод как три значения сразу, так и выводить для каждой величины отдельное приглашение. Программа должна хранить время в структурной переменной типа Time и выводить количество секунд во введенном времени. Добавьте в структуру две функции для сложения и вычитания интервалов времени.

При решении задачи учитывайте естественные допустимые значения для реализуемых характеристик.

Задание 2. Решение квадратного уравнения

Создайте структуру – решение квадратного уравнения, содержащей два вещественных поля – корни квадратного уравнения.

Реализуйте решение квадратного уравнения с помощью функции, возвращающей переменную типа структуры – решение этого уравнения с полями – корнями уравнения.

Задание 3. Решение квадратного уравнения

Чтобы вернуть из функции несколько результатов без реализации нового типа (класса или структуры), можно объединить их в кортеж. Кортеж tuple (из заголовочного файла <tuple>) допускает наличие элементов различных типов и количество объектов в кортеже должно быть известно во время компиляции.

Реализуйте решение квадратного уравнения с помощью функции, возвращающей переменную типа кортеж, содержащую решение этого уравнения (корни уравнения) и флаг, показывающий наличие корней.

Практическое занятие 8. Объявление и реализация класса. Реализация инкапсуляции. Конструкторы и деструкторы.

Упражнение 1. Реализация сущности – студент в виде класса

В этом упражнении вы создадите программу, которая будет заниматься учетом успеваемости студентов.

Студента можно представить как класс.

Конкретный студент является объектом класса со свойствами – имя (`name`), фамилия (`last_name`), средний балл(`average_score`). Также у студента есть промежуточные оценки за весь семестр. Эти оценки будут записываться в целочисленный массив из пяти элементов. После того, как все пять оценок будут проставлены, будет определен средний балл успеваемости студента за весь семестр — `average_score`.

В классе определять средний балл успеваемости будет функция (которая может выполнять какие-либо действия над данными (свойствами класса) `calculate_average_score()`.

1. Создайте новый пустой проект (**Student01**).
2. Добавьте пустой файл **Student01_main.cpp**, в котором будет находиться требуемый класс.
3. Объявите класс **Student** с указанными функциями и полями:

```
#include <iostream>
#include <string>
using namespace std;

class Student
{
public:
 // Установка имени студента
voidset_name(string student_name)
{
name = student_name;
}

// Получение имени студента
string get_name()
{
return name;
}
// Установка фамилии студента
void set_last_name(string student_last_name)
{
last_name = student_last_name;
```

```
        }

 // Получение фамилии студента
string get_last_name()
{
return last_name;
}

// Установка промежуточных оценок
void set_scores(int student_scores[])
{
for (int i = 0; i < 5; ++i) {
scores[i] = student_scores[i];
}
}

// Установка среднего балла
void set_average_score(double ball)
{
   average_score = ball;
}

// Получение среднего балла
double get_average_score()
{
   return average_score;
}

private:
     int scores[5];              // Промежуточные оценки
     double average_score;       // Средний балл
     string name;               // Имя
     string last_name;          // Фамилия
};
```

Обратите внимание, что в классе методы открыты, а свойства (поля класса) объявлены как закрытые.

Функция `set_name()` сохраняет имя студента в переменной name, а `get_name()` возвращает значение этой переменной. Принцип работы функций `set_last_name()` и `get_last_name()` аналогичен.

Функция `set_scores()` принимает массив с промежуточными оценками и сохраняет их в приватную переменную `int scores[5]`.

4. Добавьте функцию `main()`.

5. Протестируйте класс на примере создания конкретного студента. Для этого в самом начале программы создадите объект класса **Student**

```
int main()
{
// Создание объекта класса Student
Student student01;
```

Объект `student01` класса **Student** характеризует конкретного студента. Если потребуется выставить оценки всем учащимся в группе, то нужно будет создать новый объект для каждого из них.

6. После создания объекта `student01` объявите две переменные – имя и фамилию, а также массив для хранения промежуточных оценок для конкретного ученика.

```
string name;
string last_name;
int scores[5];
```

7. Реализуйте ввод имени и фамилии студента с клавиатуры:

```
// Ввод имени с клавиатуры
cout << "Name: ";
getline(cin, name);

// Ввод фамилии
cout << "Last name: ";
getline(cin, last_name);
```

8. Организуйте ввод оценок с помощью цикла, и в нем же подсчитайте сумму этих оценок:

```
// Сумма всех оценок
int sum = 0;

// Ввод промежуточных оценок
for (int i = 0; i < 5; ++i) {
        cout << "Score " << i+1 << ": ";
        cin >> scores[i];
        // суммирование
        sum += scores[i];
    }
```

9. Введенные данные передайте `set`-функциям, которые присваивают их закрытым переменным класса:

```
// Сохранение имени и фамилии в объект класса Student
    student01.set_name(name);
    student01.set_last_name(last_name);
// Сохранение промежуточных оценок в объект класса Student
   student01.set_scores(scores);
```

10. После того, как были введены промежуточные оценки, рассчитайте средний балл на основе этих оценок, а затем сохраняется это значение в закрытом свойстве `average_score`, с помощью функции `set_average_score()`:

```
 double average_score = sum / 5.0;
// Сохранение среднего балла в объект класса Student
student01.set_average_score(average_score);
```

11. Выведите на экран данные о студенте:

```
cout << "Average ball for " << student01.get_name() << " "
     << student01.get_last_name() << " is "
     << student01.get_average_score() << endl;

return 0;
}
```

12. Скомпилируйте и запустите программу, протестируйте ее работу.

Упражнение 2. Разделение реализации и представления

В этом упражнении вы отделите реализацию класса от представления.

Скопируйте для этого упражнения предыдущий проект.

1. Вынесите реализацию всех методов класса **Student**, реализованного в прошлом упражнении в отдельный файл **student.cpp**.

```
/* student.cpp */
#include <string>
#include "student.h"

// Установка имени студента
void Student::set_name(std::string student_name)
{
    Student::name = student_name;
}

// Получение имени студента
std::string Student::get_name()
{
    return Student::name;
}

// Установка фамилии студента
void Student::set_last_name(std::string student_last_name)
{
    Student::last_name = student_last_name;
}
```

```
// Получение фамилии студента
std::string Student::get_last_name()
{
    return Student::last_name;
}

// Установка промежуточных оценок
void Student::set_scores(int scores[])
{
for (inti = 0; i < 5; ++i) {
        Student::scores[i] = scores[i];
    }
}

// Установка среднего балла
void Student::set_average_score(double ball)
{
    Student::average_score = ball;
}

// Получение среднего балла
double Student::get_average_score()
{
    return Student::average_score;
}
```

2. Добавьте в проект заголовочный файл **student.h** и в нем укажите только прототипы этих методов:

```
/* student.h */
#pragma once /* Защита от двойного подключения заголовочного файла */
#include <string>
using namespace std;

class Students
{
public:
        // Установка имени студента
void set_name(string);
        // Получение имени студента
string get_name();
        // Установка фамилии студента
void set_last_name(string);
        // Получение фамилии студента
string get_last_name();
```

```
        // Установка промежуточных оценок
void set_scores(int []);
       // Установка среднего балла
void set_average_score(double);
        // Получение среднего балла
double get_average_score();

private:
        // Промежуточные оценки
int scores[5];
        // Средний балл
double average_score;
        // Имя
string name;
        // Фамилия
string last_name;
};
```

7. Создайте новый файл **main.cpp**, в котором укажите код, создающий конкретного студента:

```
#include <iostream>
#include <string>
#include "student.h"
using namespace std;

int main()
{
    // Создание объекта класса Student
    Student student01;

string name;
string last_name;

    // Ввод имени с клавиатуры
cout << "Name: ";
getline(cin, name);

    // Ввод фамилии
cout << "Last name: ";
getline(cin, last_name);

// Сохранение имени и фамилии в объект класса Student
student01.set_name(name);
    student01.set_last_name(last_name);
```

```
    // Оценки
int scores[5];
    // Сумма всех оценок
int sum = 0;

   // Ввод промежуточных оценок
for (int i = 0; i < 5; ++i) {
cout << "Score " << i+1 << ": ";
cin >> scores[i];
   // суммирование
sum += scores[i];
    }

// Сохраняем промежуточные оценки в объект класса Student
student01.set_scores(scores);
    // Считаем средний балл
double average_score = sum / 5.0;
    // Сохраняем средний балл в объект класса Student
    student01.set_average_score(average_score);
    // Выводим данные по студенту
cout<< "Average ball for " << student01.get_name() << " "
<< student01.get_last_name() << " is "
<< student01.get_average_score() <<endl;

return 0;
}
```

8. Скомпилируйте и запустите программу, протестируйте ее работу.

Упражнение 3. Создание и удаление объекта

В этом упражнении вы создадите объект, выделив для него память в куче с помощью указателя, и освободите ее после того, как закончите работу с объектом.

Работать следует с проектом прошлого упражнения.

1. В файле **main.cpp** вместо объявления объектной переменной создайте объект и выделите ему память с помощью оператора **new**:

```
/* main.cpp */
#include <iostream>
#include "student.h"

int main()
{
    // Выделение памяти для объекта Student
Student *student02 = new Student;
```

```
string name;
string last_name;

    // Ввод имени с клавиатуры
cout << "Name: ";
getline(std::cin, name);

    // Ввод фамилии
cout << "Last name: ";
getline(std::cin, last_name);


    // Оценки
int scores[5];
    // Сумма всех оценок
int sum = 0;

    // Ввод промежуточных оценок
for (int i = 0; i < 5; ++i) {
cout << "Score " << i+1 << ": ";
cin >> scores[i];
    // суммирование
sum += scores[i];
    }
```

2. Так как память для объекта выделяется посредством указателя, то для доступа к его методам и свойствам используйте оператор косвенного обращения — «->»:

```
// Сохранение имени и фамилии в объект класса Students
student02->set_name(name);
student02->set_last_name(last_name);

// Сохраняем промежуточные оценки в объект класса Student
student02->set_scores(scores);

    // Считаем средний балл
float average_score = sum / 5.0;
// Сохраняем средний балл в объект класса Student
student02->set_average_score(average_score);
    // Выводим данные по студенту
cout << "Average ball for " << student02->get_name() << " "
<< student02->get_last_name() << " is "
<< student02->get_average_score() << endl;
```

3. Освободите память, занимаемую объектом:

```
delete student02;
```

```
return 0;
}
```

4. Скомпилируйте и запустите программу, протестируйте ее работу.

Упражнение 4. Использование конструктора

В этом упражнении вы добавите в класс **Student** конструктор, который будет принимать имя и фамилию ученика, и сохранять эти значения в соответствующих переменных класса.

Работать следует с проектом прошлого упражнения.

Конструктор будет выглядеть следующим образом:

```
Student::Student(string name, string last_name)
{
    Student::set_name(name);
    Student::set_last_name(last_name);
}
```

При создании нового объекта, пользователь должен передать конструктору имя и фамилию студента, например,

```
string name = "Иван";
string last_name = "Иванов";
Student *student = new Student(name, last_name);
```

1. Добавьте прототип конструктора в файл student.h:

```
/* student.h */
#pragma once /* Защита от двойного подключения заголовочного файла */
#include <string>

class Student {
public:
// Конструктор класса Student
Student(string, string);
...
};
```

2. В файле student.cpp определите сам конструктор.

```
/* student.cpp */
#include <string>
#include "student.h"

// Конструктор Student
Student::Student(string name, string last_name)
{
    Student::set_name(name);
    Student::set_last_name(last_name);
}
```

...

3. В функции **main()** реализуйте получение от пользователя имя и фамилию студента, и сохраните их во временных локальных переменных. После этого создайте новый объект класса **Student**, передавая его конструктору эти переменные.

```
/* main.cpp */
#include <iostream>
#include <string>
#include "student.h"
using namespace std;

int main()
{
string name;
string last_name;

// Ввод имени с клавиатуры
cout << "Name: ";
getline(cin, name);

    // Ввод фамилии
cout<< "Last name: ";
getline(cin, last_name);

// Передача параметров конструктору
Student *student02 = new Student(name, last_name);

// Оценки
int scores[5];
    // Сумма всех оценок
int sum = 0;

// Ввод промежуточных оценок
for (int i = 0; i < 5; ++i) {
cout << "Score " << i+1 << ": ";
cin >> scores[i];
// суммирование
sum += scores[i];
    }

     // Сохраняем промежуточные оценки в объект класса Student
student02->set_scores(scores);
```

```
    // Считаем средний балл
double average_score = sum / 5.0;
    // Сохраняем средний балл в объект класса Student
    student02->set_average_score(average_score);
    // Выводим данные по студенту
cout << "Average ball for " << student02->get_name() << " "
<< student02->get_last_name() << " is "
<< student02->get_average_score() << endl;
// Удаление объекта student из памяти
delete student02;

return 0;
}
```

4. Постройте и запустите программу. Протестируйте ее работу.

Упражнение 5. Сохранение данных в файл

В этом упражнении вы реализуете возможность сохранения данных после окончания работы с программой: запись данных в текстовый файл. Это вы сделаете в соответствии с RAII (Resource Acquisition Is Initialization) — одним из ключевых принципов эффективного и безопасного программирования в C++, особенно при работе с ресурсами (память, файлы, потоки и т.п.). RAII означает, что ресурсы привязываются к времени жизни объектов. Ресурс (например, указатель на память или дескриптор файла) захватывается в конструкторе объекта и освобождается в деструкторе.

Требуется сохранить данные о студентах в следующем виде: оценки каждого студента будут находиться в отдельной строке, имя и фамилии будут разделяться пробелами. После имени и фамилии ученика ставится еще один пробел, а затем перечисляются все его оценки. Например,

```
Василий Федоров 5 4 5 3 3
Иван Сидоров 5 5 3 4 5
Андрей Иванов 5 3 3 3 3
```

Для работы с файлами необходимо воспользоваться библиотекой **fstream**, которая подключается в заголовочном файле следующим образом:

```
#include <fstream>
```

Сохранить данные можно с помощью метода, в котором реализовать логику работы с **fstream**:

```
// Запись данных о студенте в файл
void Student::save()
{
ofstream fout("students.txt", ios::app);
```

```
fout << Student::get_name() << " "
<< Student::get_last_name() << " ";

for (int i = 0; i <5; ++i) {
fout << Student::scores[i] <<" ";
    }

fout << endl;
fout.close();
}
```

Переменная **fout** — это объект класса **ofstream**, который входит в состав библиотеки **fstream**. Класс **ofstream** используется для записи каких-либо данных во внешний файл. Конструктор этого класса принимает в качестве параметров имя выходного файла и режим записи. В данном случае, используется режим добавления — **ios:app**. После завершения работы с файлом, необходимо вызвать метод **close**() для того, чтобы закрыть файловый дескриптор.

Для сохранения оценок студента надо будет вызывать созданный метод **save**(), например:

```
Student neo->save();
```

Теперь необходимо решить, когда будет вызываться метод, сохраняющий данные. Можно сохранять все оценки после того, как работа с объектом студент закончена. Для этого нужно будет создать деструктор класса **Student**, который будет вызывать метод `save()` перед уничтожением объекта (обратите внимание, что ресурс захватывается при инициализации, и освобождается при уничтожении).

```
// Деструктор Student
Student::~Student()
{
    Student::save();
}
```

1. Добавьте прототипы деструктора и метода `save()` в файл student.h:

```
/* student.h */
#pragma once /* Защита от двойного подключения заголовочного файла */
#include <string>

class Student {
    public:
    // Запись данных о студенте в файл
        void save();
    // Деструктор класса Student
       ~Student();

...
```

```
};
```

2. Определите эти функции в файле реализации students.cpp.

```
/* student.cpp */
#include <string>
#include <fstream>

#include "student.h"

// Деструктор Student
Student::~Student()
{
    Student::save();
}

// Запись данных о студенте в файл
void Student::save()
{
    ofstream fout("students.txt", ios::app);

fout << Student::get_name() <<" "
<< Student::get_last_name() <<" ";

for (int i = 0; i <5; ++i) {
fout << Student::scores[i] <<" ";
    }

fout << endl;
fout.close();
}

...
```

3. Содержимое файла main.cpp оставьте без изменений.
4. При необходимости сохранять русский текст русифицируйте консоль (практическое занятие 1, упр.2).
5. Скомпилируйте и запустите программу. Перед тем, как приложение завершит свою работу, будет создан новый текстовый файл с оценками – students.txt.

Контрольные задания

Задание 1. Класс Time

Создайте класс с именем **Time**, содержащий три поля типа **int**, предназначенные для хранения часов, минут и секунд.

Один из конструкторов класса должен инициализировать поля нулевыми значениями (примените для этого списки инициализации), а другой конструктор — заданным набором значений (вариант на ваше усмотрение).

В конструктор с параметрами добавьте реализацию приведения больших неправильных значений параметров к правильным значениям, например, если клиент захочет создать время с параметром минут равным 70, то конструктор должен преобразовать это значение к правильному виду: количество часов и минут.

Создайте функцию-член класса, которая будет выводить значения полей на экран в формате 11:59:59, и функцию-член класса, складывающую значения двух объектов типа **Time** – подумайте, как наиболее оптимально это реализовать (сколько передавать параметров в качестве аргументов и каким образом вернуть сумму).

Где это возможно и оправдано, сделайте методы константными.

В функции `main()` следует создать два инициализированных объекта (подумайте, должны ли они быть константными) и один неинициализированный объект. Затем сложите два инициализированных значения, а результат присвойте третьему объекту и выведите его значение на экран.

Практическое занятие 9. Обработка исключительных операций

Упражнение 1. Безопасная функция деления

В этом упражнении вы реализуете обработку исключительной ситуации при делении на нуль.

1. Создайте новое приложение и добавьте файл исходного кода.
2. До функции `main()` объявите класс `DivideByZeroError`, представляющий ошибку, возникающую при делении на нуль::

```
#include <iostream>
#include <string>

using namespace std;

class DivideByZeroError
{
public:
     DivideByZeroError  (): message  ("Деление на нуль")  {  }
     void printMessage  ()  const {cout << message << endl;}
private:
     string message;
};
```

3. Определите функцию quotient, реализующей операцию деления. Укажите проверку на равенство нулю и сгенерируйте соответствующее исключение:

```
float quotient (int num1, int num2)
{
    if (num2 == 0)
     throw DivideByZeroError();
    return  (float)num1 / num2;
}
```

4. В функции main() запросите у пользователя два числа и поместите вызов функции quotient в блок `try`:

```
int main ()
{
 cout << "Введите два целых числа для расчета их частного:\n";
 int number1,  number2;
 cin >> number1;
 cin >> number2;
 try
 {
     float result = quotient(number1,  number2);
     cout << "Частное равно " << result << endl;
 }
```

5. Далее в блоке `catch` обработайте возможную ошибку указанного типа, передав объект класса-исключения по ссылке (это предотвратит создание копии исключения компилятором, что является затратной операцией):

```
 catch (DivideByZeroError& error)
 {
     cout <<  "ОШИБКА: ";
     error.printMessage();
      return 1;        // завершение программы при ошибке
 }
return 0;             // нормальное завершение программы
}
```

6. Скомпилируйте и запустите программу. Введите значение знаменателя, равного нулю для проверки реакции программы на эту ситуацию.

Упражнение 2. Безопасное деление в цикле

В этом упражнении вы реализуете безопасный вызов функции в цикле. В случае, когда параметр функции примет “опасное” значение блок try перехватит исключение, передаст его обработчику и затем программа продолжит свое выполнение.

1. В функции main() исправьте код запроса от пользователя двух чисел на одно – числитель операции деления.
2. Блок-try поместите в цикл for, реализующий вызов функции на интервале изменения знаменателя, например, от -10 до 10:

```
for (int i = -10; i < 10; i++)
{
	try
	{
		float result = quotient(number1,  i);
		cout << "Частное равно " << result << endl;
	}
	catch (DivideByZeroError& error)
	{
		cout <<  "ОШИБКА: ";
		error.printMessage();
	}
}
```

3. Скомпилируйте и запустите программу. Введите значение числителя и просмотрите вывод результатов деления.

Упражнение 3. Реализация исключения с параметрами

В этом упражнении вы реализуете обработку исключительной ситуации, при этом выполните не только передачу управления обработчику, но и создадите объект класса исключения с помощью вызова его конструктора. Обработчик исключений затем сможет извлечь данные из этого объекта при перехвате исключения.

1. Откройте приложение с файлом исходного кода Student01_main.cpp – воспользуйтесь решением упражнений 1 практического занятия 8.
2. Добавьте в класс `Student` класс исключений `ExScore`, который будет использоваться для связывания выражения генерации исключения с блоком-обработчиком.
3. В классе исключения объявите первую переменную для имени функции, в которой возникает ошибка и вторую – для хранения ошибочного значения.
4. В классе исключения объявите конструктор с двумя параметрами для инициализации свойств объекта исключения.

В итоге класс исключений должен иметь следующий вид:

```
public:
	class ExScore    //класс исключений
	{
	public:
		string origin;   //для имени функции
```

```
        int iValue;      //для хранения ошибочного значения

        ExScore(string or, int sc)
        {
              origin = or;     //строка с именем виновника ошибки
              iValue = sc;     //сохраненное неправильное значение
        }
    };
```

5. Ошибка может возникнуть при передаче массива оценок, введенных пользователем в поле массив оценок класса `Student`. Поэтому добавьте в функцию `set_scores` проверку того, что оценки не превышают значения 5, и в случае наступления этого события укажите выражение генерации объекта исключения:

```
void set_scores(int student_scores[])
{
   for (int i = 0; i < 5; ++i) {
          if(student_scores[i] > 5)
          throw ExScore("в функции set_scores()", student_scores[i]);
          scores[i] = student_scores[i];
}
   }
```

6. В методе `main()` потенциально опасным кодом является сохранение промежуточных оценок в объект класса `Student`, поэтому поместите этот код (а также расчет среднего значения и вывод информации на экран) в блок `try`:

```
try
   {
         student01.set_scores(scores);
        ...
   }
```

7. После этого добавьте обработчик ошибки и, используя свойства объекта исключения, получите информацию об ошибке:

```
catch(Student::ExScore& ex)
 {
      cout << "\nОшибка инициализации " << ex.origin;
      cout << "\nВведенное значение оценки "  << ex.iValue <<
"является недопустимым\n";
 }
```

8. Постройте и запустите приложение. Введите пять оценок и хотя бы одну из них больше пяти. В этом случае при записи массива оценок функцией `set_scores` в поле класса возникнет ошибка, которая будет перехвачена и обработана.

Контрольные задания

Задание 1. Безопасная реализация класса Triangle

Требуется разработать класс **Triangle**, представляющий треугольник, который задается тремя сторонами.

Для класса определить функцию, вычисляющую площадь треугольника по трем сторонам (см. практическое занятие 1).

Реализовать генерацию исключительной ситуации при попытке задать стороны недопустимой длины – если хотя бы одна из сторон имеет длину большую, чем сумма двух других сторон.

Задание 2. Безопасная реализация класса Time

Добавьте в класс **Time** класс, реализующий возможность реагировать на исключительные ситуации, возникающие, например, при создании «неправильных» объектов, проведении операций с объектами класса и т.д.

Практическое занятие 10. Реализация отношений между классами

Упражнение 1. Отношение ассоциации

В этом упражнении вы реализуете отношение ассоциации. Добавите класс **IdCard**, представляющий идентификационную карточку студента. Каждому студенту может соответствовать только одна идентификационная карточка, таким образом, мощность связи 1 к 1.

1. Откройте приложение упражнения 3 занятия 8.
2. Добавьте в проект новый файл IdCard.h и объявите новый класс **IdCard,** который содержит два закрытых поля – номер карточки и статус (категорию), а также соответствующие методы доступа к этим полям и конструкторы:

```
#pragma once
#include <string>
using namespace std;

class IdCard
{
private:
      int number;
      string category;

public:
      IdCard();
      IdCard(int);
      IdCard(int, string);
      void setNumber(int newNumber);
      int getNumber();
```

```
        void setCategory(string cat);
        string getCategory();
    };
```

3. Добавьте файл реализации класса IdCard.cpp и определите в нем конструкторы и функции-члены класса **IdCard**:

```
 #include "IdCard.h"
IdCard::IdCard(int n)
{
      number = n;
      category = «Не установлена»;
}
IdCard::IdCard()
{
      number = 0;
      category = "Не установлена";
 }
IdCard::IdCard(int n, string cat)
{
      number = n;
      category = cat;
 }

void IdCard::setNumber(int newNumber)
{
    number = newNumber;
}

int IdCard::getNumber()
{
   return number;
}

void IdCard::setCategory(string cat)
{
      category = cat;
}

string IdCard::getCategory()
{
      return category;
}
```

4. В классе `Student` (файл Student.h) объявите указатель **iCard** на объект типа **IdCard:**

```
IdCard* iCard;
```

и методы для управления доступом к этому полю (записи и получения значения):

```
void setIdCard(IdCard *c);
IdCard getIdCard();
```

5. В файле Student.cpp внесите изменения в конструктор и реализуйте новые методы:

```
Student::Student(string name, string last_name, IdCard *id)
{
Student::set_name(name);
Student::set_last_name(last_name);
Student::setIdCard(id);
}
void Student::setIdCard(IdCard* c)
{
      iCard = c;
}

IdCard Student::getIdCard()
{
      return *iCard;
}
```

6. В функции `main()` создайте объект класса **IdCard** с передачей двух значений для инициализации объекта:

```
IdCard idc(123, "Базовый");
```

7. При создании объекта класса **Student** добавьте ему новый параметр для инициализации:

```
Student *student02 = new Student(name, last_name, idc);
```

8. Добавьте вывод новых данных о студенте:

```
cout << "IdCard: " << student02->getIdCard().getNumber() << endl;
cout << "Category: " << student02->getIdCard().getCategory() << endl;
```

9. Постройте и протестируйте приложение. Проверьте вывод новых данных.
10. Самостоятельно реализуйте запрос данных о номере карты и статусе с клавиатуры. Для доступа к закрытым полям класса используйте соответствующие функции-члены класса.
11. Постройте и протестируйте приложение.

Упражнение 2. Отношение композиции

В первом упражнении отношение ассоциации было реализовано как отношение агрегации, когда объект целого содержит указатели или ссылки на части, но не владеет ими, части могут жить сами по себе и использоваться разными целыми и части могут существовать независимо от целого.

В этом упражнении вы реализуете отношение между сущностями студента и его карточки как отношение композиции, когда объект целого содержит в себе объекты частей как поля, части создаются и уничтожаются вместе с целым и без целого части существовать не могут.

1. В классе `Student` (файл Student.h) объявите **iCard** как поле типа **IdCard:**

```
IdCard iCard;
```

и методы для управления доступом к этому полю (записи и получения значения):

```
void setIdCard(int id, string c);
int getIdCard();
```

и добавьте (если не было) метод вывода информации о студенте:

```
void display();
```

2. В файле Student.cpp внесите изменения в конструктор и реализуйте новые методы:

```
Student::Student(string name, string last_name, int id,
string cat)
{
Student::set_name(name);
Student::set_last_name(last_name);
Student::setIdCard(id, cat);
}
void Student::setIdCard(int id, string c)
{
      iCard = IdCard(id, c);
}

int Student::getIdCard()
{
      return iCard.getNumber();
}
void Student::display()  // вывод всех данных о студенте
 {
      cout << last_name << " " << name << "\t" << average_score <<
"\t" << iCard.getNumber()<< "\t" << iCard.getCategory() << endl;
}
```

3. В функции `main()` теперь не требуется создавать объект класса **IdCard** – он будет создан внутри класса студента (смысл композиции), удалите код создания объектов-карточек.
4. А вот при создании объекта класса **Student** добавьте ему новые параметры для инициализации:

```
Student *student02 = new Student(name, last_name, 101, "base");
```

5. И в конце замените вывод данных о студенте на вызов нового метода:

```
student02->display();
```

6. Постройте и протестируйте приложение. Проверьте вывод новых данных.
7. Обратите внимание, что класс `IdCard` не был изменен, ему «все равно» как его будет использовать класс целое (студент).

Контрольные задания

Задание 1. Реализация класса Triangle

В этом задании требуется создать класс **Triangle**, определяемый тремя точками – объектами соответсвующего класса **Dot.**

Элементы класса **Triangle**:

- Три точки – объекты класса **Dot**.
- Конструктор.
- Методы, позволяющие:
  - вывести длины сторон треугольника;
  - расчитать периметр треугольника;
  - расчитать площадь треугольника

Класс **Dot:**

Определяется двумя координатами и функцией – расстоянием между точками.

Файл dot.h

```
class Dot
{
private:
     double x;
     double y;
public:
     Dot();
     Dot(double x, double y);
     double distanceTo(Dot point);
};
```

Файл dot.cpp

```
#include "dot.h"
#include <math.h>

Dot::Dot()
{
     x = 0; y = 0;
}

Dot::Dot(double x, double y)
{
     this -> x = x;
```

```
    this -> y = y;
}

double Dot::distanceTo(Dot point)
{
    return sqrt(pow(point.x - x,2) + pow(point.y - y,2));
}
```

Реализуйте два варианта отношений между треугольником (целое) и точкой (часть) – как композиция (целое отвечает за жизненный цикл части) и как агрегация (часть и целое могут существовать по-отдельности).

Практическое занятие 11. Перегрузка операций

Упражнение 1. Перегрузка бинарных операций

В этом упражнении вы изучите класс, моделирующий расстояния, выраженные в английской системе мер. В этот класс вы добавите возможность выполнять основные арифметические операции над объектами этого класса.

1. Создайте новый проект и добавьте файл исходного кода.
2. Изучите класс `Distance` и добавьте его в файл (обратите внимание на объявление операторной функции **оператор+,** которая перегружает оператор сложения):

```
class Distance
{
private:
    int feet;
    float inches;
public:
    // конструктор по умолчанию
    Distance ( ) : feet (0), inches (0.0) { }
// конструктор с двумя параметрами
Distance (int ft, float in) : feet (ft), inches (in) { }

    void getdist()
    {
        cout <<"\nВведите число футов: ";
        cin >> feet;
        cout <<"\nВведите число дюймов: ";
        cin >> inches;
    }
    void showdist()
    {
        cout << feet << "\'-" << inches << "\"\n";
```

```
    }
    Distance operator+ (const Distance&) const;
};
```

3. Определите операторную функцию сложения двух расстояний:

```
Distance Distance::operator+ (const Distance& d2) const
{
   int f = feet + d2.feet;
   float i = inches + d2.inches;
   if (i>= 12.0)
   {
     i -= 12.0;
     f++;
   }
   return Distance (f, i);
}
```

4. В методе `main()` определите четыре переменных типа `Distance`:
   `Distance dist1, dist2, dist3. Dist4;`
5. Присвойте первым двум переменным значения с помощью функции `getdist();`
6. Проверьте работу перегруженного оператора сложения:
   `dist3 = dist1 + dist2;`
7. Проверьте работу перегруженного оператора сложения при выполнении цепочки операций:
   `dist4 = dist1 + dist2 + dist3;`
8. Реализуйте отображение полученных значений на экран с помощью функции `showdist()`, например,
   ```
   cout << "\ndist1 = ";
   dist1.showdist();
   ```
9. Постройте и протестируйте приложение.
10. Добавьте в класс перегруженную операцию вычитания (реализуйте ее с помощью дружественной функции), которая вычисляет разность двух расстояний. Она должна позволять выполнение выражений типа
    `dist3 = dist1 – dist2;`

При реализации оператора вычитания учтите, что эта операция никогда не будет использоваться для вычитания большего расстояния из меньшего (так как отрицательного расстояния быть не может).

11. Для замены функции showdist() перегрузите оператор потокового вывода <<. Перегрузка оператора вывода << аналогична перегрузке арифметических операторов (они являются бинарными операторами), за исключением того, что их типы различны и функция может быть только дружественной:

a. Объявите в классе перегружаемую функцию дружественной (обратите внимание, что левый параметр возвращается в качестве ссылки, это предотвращает создание копии std::ostream, так как std::ostream запрещает своё копирование):

```
    friend std::ostream& operator<< (std::ostream &out, const Distance
&dist);
```

b. Теперь реализуйте объявленную функцию (интерфейс вывода остается, как и у функции showdist):

```
std::ostream& operator<< (std::ostream &out, const Distance &dist)
{
    out << dist.feet << "\'-" << dist.inches << "\"\n";
    return out;
}
```

12. Реализуйте отображение полученных значений на экран не с помощью функции `showdist()`, а с помощью перегруженного оператора:

```
cout << "\ndist1 = " << dist1;
```

13. Постройте и протестируйте приложение.

Упражнение 2. Преобразования объектов в основные типы и наоборот

В этом упражнении вы выполните преобразование определенного пользователем типа (класс `Distance`) в основной тип (`float`) и наоборот. Так как компилятору ничего не известно (кроме того, что написал сам разработчик) об определенных пользователем типах, вы должны сами написать функции для преобразования типов. Для перехода от основного типа (в нашем случае `float`) к определенному пользователем типу, такому, как `Distance`, вы используете конструктор с одним аргументом.

1. В класс `Distance` добавьте константу – коэффициент перевода метров в футы:

```
const float MTF;
```

2. В конструкторы добавьте инициализацию нового поля класса – константы:

```
Distance ( ) : feet (0), inches (0.0), MTF (3.280833F) { }
Distance (int ft, float in) : feet (ft), inches (in), MTF (3.280833F)
 { }
```

3. Для преобразования вещественного типа в тип `Distance` добавьте конструктор, который будет вызываться при создании объекта с передачей ему параметра – значения вещественного типа:

```
Distance (float meters) : MTF (3.280833F)
{
   float fltfeet = MTF * meters;       // перевод в футы
   feet = int (fltfeet);               // число полных футов
```

```
    inches = 12 * (fltfeet-feet);       // остаток – это дюймы
}
```

Конструктор предполагает, что аргумент представляет собой метры. Он преобразовывает аргумент в футы и дюймы и присваивает полученное значение объекту. Таким образом, преобразование от метров к переменной типа `Distance` будет выполняться вместе с созданием объекта в строке `Distance dist1 = 2.25;`

4. Добавьте в класс операторную функцию, которая обеспечит прием значения объекта класса `Distance`, преобразовывает его в значение типа `float`, представляющее собой метры, и возвращает это значение

```
operator float( ) const
{
  float fracfeet = inches / 12;
  fracfeet += static_cast<float>( feet );
  return fracfeet / MTF;
}
```

5. В методе `main()` создайте объект класса и присвойте ему значение вещественного типа:

```
Distance dist1 = 2.35F;
```

Для инициализации объекта используется конструктор, переводящий метры в футы и дюймы.

6. Объявите переменную вещественного типа:

```
float mtrs;
```

7. Присвойте значение переменной, используя оператор перевода в метры:

```
mtrs = static_cast<float>(dist1);
```

8. Проверьте неявное приведение типа:

```
mtrs = dist2;
```

9. Реализуйте вывод значения переменной на экран.
10. Постройте и протестируйте приложение.

Упражнение 3. Перегрузка операций операндов различных типов

В первом упражнении вы научили складываться (вычитаться) операнды типа Distance, во втором упражнении вы реализовали преобразование вещественного типа к типу Distance.

В этом упражнении реализуйте перегрузку бинарных операторов сложения и вычитания для работы с операндами разных типов, например, сложения (вычитания) объекта расстояния и вещественного значения, для этого нужно написать две функции для операции, например, `dist + 2,8` и `2,8 + dist` по одной на каждый случай.

Упражнение 4. Перегрузка оператора индексации

В этом упражнении вы реализуете перегрузку оператора индексации. Оператор индексации является одним из операторов, перегрузка которого должна выполняться через функцию-член класса. Функция перегрузки оператора [ ] всегда будет принимать один параметр: значение индекса (элемент, к которому требуется доступ).

В этом упражнении вы напишите программу, которая позволит присваивать оценки ученикам, указывая только имя ученика. Для этого будет создан класс, реализующий эту функциональность, по сути, это будет контейнер map (в контейнере map все элементы хранятся в виде пары ключ-значение, таким контейнерам посвящено занятие № 15): имя ученика – ключ, оценка (тип char) — значение. Ключ должен быть уникальным и использоваться для доступа к связанной паре.

1. Создайте новый проект и добавьте файл исходного кода.
2. Объявите структуру StudentGrade с двумя элементами: имя студента (std::string) и оценка (char):

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

struct StudentGrade
{
    string name;
    char grade;
};
```

3. Добавьте объявление класса GradeMap:
    a. Для реализации класса как контейнера укажите в поле класса std::vector типа StudentGrade с именем m_map (получится класс-контейнер на основе вектора).
    b. Добавьте пустой конструктор по умолчанию.
    c. Объявите функцию (прототип) перегрузки оператора [ ] для этого класса, эта функция перегрузки должна принимать параметр std::string (имя ученика) и возвращать ссылку на его оценку

В результате класс должен выглядеть следующим образом:

```
class GradeMap
{
private:
    vector<StudentGrade> m_map;
public:
```

```
    GradeMap()
    { }
    char& operator[](const string &name);
};
```

4. Реализуйте объявленную функцию перегрузки оператора [ ] для этого класса. Функция должна принимать параметр std::string (имя ученика) и возвращать ссылку на его оценку:
    a. Сначала выполните поиск указанного имени ученика в векторе (используйте цикл foreach) – если ученик нашёлся, то верните из функции ссылку на его оценку.
    b. В противном случае (если студент не найден) для добавления StudentGrade для нового ученика используйте функцию std::vector::push_back().
    c. Далее нужно будет возвратить ссылку на оценку студента, который был только что добавлен в std::vector — для этого используйте std::vector::back().

В результате вид функции перегрузки должен быть таким:

```
char& GradeMap::operator[](const string &name)
{
    // Найдём ли мы имя ученика в векторе
    for (auto &ref : m_map)
    {
        // Если нашли, то возвращаем ссылку на его оценку
        if (ref.name == name)
            return ref.grade;
    }
    // Не нашли - создаём новый StudentGrade для нового ученика
    StudentGrade temp{ name, ' ' };
    // Помещаем его в конец вектора
    m_map.push_back(temp);
    // И возвращаем ссылку на его оценку
    return m_map.back().grade;
}
```

5. В функции main() создайте объект-коллекцию класса GradeMap, запишите в него, используя перегруженный оператор индекса, произвольные данные и выведите их на экран. Затем запросите новое имя и оценку и запишите эти данные в объект-коллекцию, выведите новые данные на экран:

```
int main()
{
    GradeMap grades;
    grades["John"] = 'A';
    grades["Martin"] = 'B';
```

```
    cout << "John has a grade of " << grades["John"] << endl;
    cout << "Martin has a grade of " << grades["Martin"] << endl;

    cout << "New name and grade?" << endl;
    string name;
    char grade;
    cin >> name >> grade;
    grades[name] = grade;
    std::cout << name << " has a grade of " << grades[name] << endl;

    return 0;
}
```

6. Постройте и протестируйте приложение. Обратите внимание на работу перегруженного оператора индекса [ ].

Контрольные задания

Задание 1. Перегрузка операторов в классе Time

Требуется реализовать в классе **Time** (добавить в ранее созданный класс) перегрузку следующий операций:

- сложение объектов Time,
- вычитание объектов Time,
- сложение объекта Time и переменной вещественного типа,
- сложение переменной вещественного типа и объект Time,
- сравнение двух объектов Time.

Реализуйте возможность преобразования полученных результатов к корректному виду.

В функции main() создайте требуемые объекты и проверьте работу перегруженных операций.

Задание 2. Реализация сортировки точек в векторе

Разработайте класс `Point` с двумя полями вещественного типа – x и y и конструктором с параметрами, а также с функцией расчета расстояния до центра координат.

Реализуйте для класса `Point` перегрузку операторов *передать в поток вывода* (<<) и *сравнения* (<) для того, чтобы следующий фрагмент кода работал (используется стандартный алгоритм сортировки):

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>            //  для алгоритма сортировки
```

```
int main()
{
std::vector<Point> v;
v.push_back(Point(1,2));
v.push_back(Point(10,12));
v.push_back(Point(21,7));
v.push_back(Point(4,8));
std::sort(v.begin(), v.end()); // требуется перегрузка оператора < для
                               // класса Point
for (auto &point : v)
std::cout << point << '\n'; // требуется перегрузка оператора << для
                            // класса Point
return 0;
}
```

Указание: при перегрузке оператора сравнения сравнивайте по расстоянию до центра координат, тогда сортировка вектора точек будет проходить по этому признаку.

Практическое занятие 12. Реализация наследования

Наследование позволяет реализовать иерархические отношения между классами в задачах, в которых можно выделить общие свойства и поведение, например, в базе данных ВУЗа должна храниться информация обо всех студентах и преподавателях. Представить все данные в одном классе не получится, поскольку для преподавателей понадобится хранить данные, которые для студента не применимы, и наоборот.

Упражнение 1. Создание иерархии классов

В этом упражнении вы создадите базовый класс **human**, который будет описывать модель человека (в нем будут храниться имя, фамилия и отчество) и производный от него класс.

1. Создайте новый проект – **SchoolCpp**.
2. Создайте файл human.h.
3. Реализуйте класс, объявите в поле класса свойства (имя, фамилия и отчество), конструктор с тремя параметрами для инициализации этих свойств и методы получения значений в поля класса:

```
// human.h
#include <string>
#include <sstream>
#pragma once /* Защита от двойного подключения заголовочного файла */
class human {
    public:
```

```
        // Конструктор класса human
        human(std::string last_name, std::string name, std::string
second_name)
        {
            this->last_name = last_name;
            this->name = name;
            this->second_name = second_name;
        }

        // Получение ФИО человека
        std::string get_full_name()
        {
            std::ostringstream full_name;
            full_name << this->last_name << " "
                << this->name << " "
                << this->second_name;
            return full_name.str();
        }

    private:
        std::string name; // имя
        std::string last_name; // фамилия
        std::string second_name; // отчество
};
```

В программе используется класс `string stream` (библиотека **IOStream**), который позволяет связать поток ввода-вывода со строкой в памяти. Всё, что выводится в такой поток, добавляется в конец строки; всё, что считывается из потока — извлекается из начала строки.

4. Создайте новый класс **student**, который будет наследником класса **human**. Поместите его в файл student.h. Обратите внимание, что вместо массива для хранения оценок используется вектор.

```
// student.h

#include "human.h"
#include <string>
#include <vector>

class student : public human {
    public:
        // Конструктор класса Student
        student(std::string last_name, std::string name, std::string
second_name,
```

```
            std::vector<int> scores) : human(last_name, name,
second_name) {
            this->scores = scores;
        }

        // Получение среднего балла студента
        float get_average_score()
        {
            // Общее количество оценок
            unsigned int count_scores = this->scores.size();
            // Сумма всех оценок студента
            unsigned int sum_scores = 0;
            // Средний балл
            float average_score;

            for (unsigned int i = 0; i < count_scores; ++i) {
                sum_scores += this->scores[i];
            }

            average_score = (float) sum_scores / (float) count_scores;
            return average_score;
        }

    private:
        // Оценки студента
        std::vector<int> scores;
};
```

Функция `get_average_score()` вычисляет среднее арифметическое всех оценок студента. Все публичные свойства и методы класса **human** будут доступны в классе **student**.

5. Постройте и протестируйте приложение.

Упражнение 2. Создание объекта класса student

В этом упражнении вы протестируйте работу класса **student**, в файле main.cpp создадите объект класса, сохраняя в нем его имя, фамилию, отчество и список оценок.

1. В файле main.cpp для работы с классом **student** создайте вектор оценок и добавьте произвольные оценки:

```
// main.cpp

#include <iostream>
#include <vector>
```

```
#include “human.h”
#include “student.h”

int main()
{
// Оценкистудента
std::vector<int> scores;

// Добавление оценок студента в вектор
scores.push_back(5);
scores.push_back(3);
scores.push_back(4);
scores.push_back(4);
scores.push_back(5);
scores.push_back(3);
scores.push_back(3);
scores.push_back(3);
    scores.push_back(3);
```

2. Создайте конкретного студента – объект класса **student**:

```
student *stud = new student(“Петров”, “Иван”, “Алексеевич”, scores);
```

3. Выведете на экран полное имя студента, используя унаследованный метод класса **human**

```
std::cout<< stud->get_full_name() <<std::endl;
```

После инициализации объекта, происходит вывод полного имени студента с помощью функции `get_full_name`. Эта функция была унаследована от базового класса **human.**

4. Реализуйте вывод среднего балла студента с помощью функции `get_average_score` класса **student**:

```
std::cout << ”Средний балл: “<< stud->get_average_score() <<std::endl;

return 0;
}
```

5. Постройте и протестируйте приложение.

Упражнение 3. Работа с классом teacher

В этом упражнении вы создадите еще один класс (**teacher**), в котором будут храниться данные преподавателей. Общие свойства с классом **human** будут от него унаследованы, и появится новое свойство – количество учебных часов, отведенное преподавателю на единицу времени (семестр).

1. Создайте файл teacher.h.
2. Объявите класс **teacher** производным от класса **human.**

3. Объявите свойство – количество учебных часов, метод возвращающий это значение и конструктор класса

```
// teacher.h

#include "human.h"
#include <string>

class teacher : public human {
    // Конструктор класса teacher
    public:
        teacher(
            std::string last_name,
            std::string name,
            std::string second_name,
            // Количество учебных часов за семестр у преподавателя
            unsigned int work_time
        ) : human(
            last_name,
            name,
            second_name
        ) {
            this->work_time = work_time;
        }

        // Получение количества учебных часов
        unsigned int get_work_time()
        {
            return this->work_time;
        }

    private:
        // Учебные часы
        unsigned int work_time;
};
```

4. В файле main.cpp создайте объекта класса **teacher** и протестируйте его работу:

```
#include <iostream>
#include “human.h”
#include “teacher.h”

int main()
{
```

```
...
    unsigned int teacher_work_time = 40;

    teacher *tch = new teacher("Сергеев", "Дмитрий", "Сергеевич",
teacher_work_time);

    std::cout << tch->get_full_name() << std::endl;
    std::cout << "Количество часов: " << tch->get_work_time() <<
std::endl;

    return 0;
}
```

5. Постройте и протестируйте приложение.

Практическое занятие 13. Применение полиморфизма

Упражнение 1. Реализация полиморфного вызова

В этом упражнении вы создадите структуру классов и реализуете полиморфный вызов функций производных классов. Система классов включает базовый класс `Item`, хранящий название и цену единицы хранения, и два класса: `Paperbook`, в котором происходит учет количества страниц, и `AudioBook`, в котором происходит учет минут звучания аудиофайла. Каждый из классов имеет метод `getdata()`, запрашивающий информацию у пользователя, и `putdata()` для вывода данных на экран.

1. Создайте новое приложение и добавьте файл исходного кода.
2. Объявите класс единицы хранения `Item`, хранящий название и цену единицы хранения:

```
#include <iostream>
#include <string>
#include <windows.h>
using namespace std;

class Item
{
private:
     string title;
     double price;
```

3. Добавьте открытые виртуальные методы: `getdata()`, запрашивающий информацию у пользователя и `putdata()`, выводящий данные на экран:

```
public:
     virtual void getdata()
     {
          cout << "\nВведите заголовок: ";
          cin >> title;
```

```
            cout << "Введите цену: ";
            cin >> price;
      }
      virtual void putdata()
      {
            cout << "\nЗаголовок: " << title;
            cout << "\nЦена:" << price;
      }
};
```

4. Добавьте класс `Paperbook`, (как производный от базового класса `Item`) в котором объявите переменную для учета количества страниц и переопределите методы базового класса. В этих методах добавьте код для ввода количества страниц и вывода информации о бумажных книгах.

```
class Paperbook: public Item
{
private:
            int pages;
public:
            void getdata()
            {
                  Item::getdata();
                  cout << "Введите число страниц:";
                  cin >> pages;
            }
            void putdata()
            {
                  Item::putdata();
                  cout << "\nСтраниц:" << pages;
            }
};
```

5. Добавьте класс `Audiobook`, (как производный от базового класса `Item`) в котором объявите переменную для учета времени звучания и переопределите методы базового класса. В этих методах добавьте код для ввода времени звучания и вывода информации об аудиокнигах.

```
class AudioBook: public Item
{
private:
            double time;
public:
            void getdata()
            {
            Item::getdata();
            cout << "Введите время звучания:";
            cin >> time;
```

```
        }
        void putdata()
        {
        Item::putdata();
        cout << "\nВремя звучания:" << time;
        }
};
```

6. В функции `main()` создайте массив указателей на класс `Item`.

```
int main()
{
  SetConsoleOutputCP(1251);
  Item* pubarr [100];
```

7. В цикле запросите у пользователя данные о выборе варианта заполнения данных: бумажная книга или аудиофайл, затем с помощью оператора `new`создайте новый объект классов `Paperbook` или `AudioBook`.

```
int n = 0;
char choice;
do
 {
cout << "\nВводить данные для книги или звукового файла (b/a)?";
      cin >> choice;
      if(choice == 'b')
            pubarr[n] = new Paperbook;
      else
            pubarr[n] = new AudioBook;
      pubarr[n++]->getdata();
      cout << "Продолжать((y/n)?";
      cin >> choice;
}
while(choice == 'y');
```

8. Реализуйте вывод данных о заполненных единицах хранения с помощью цикла `for`:

```
for(int j=0; j<n; j++)              //цикл по всем объектам
      pubarr[j]->putdata();        //вывести данные о публикации
cout << endl;

return 0;
}
```

9. Когда пользователь заканчивает ввод исходных данных, выводится результат для всех введенных книг и файлов.
10. Постройте и протестируйте приложение.

Контрольные задания

Задание 1. Иерархия транспортных средств

В этом задании требуется использовать виртуальные функции и абстрактные классы для реализации полиморфного поведения объектов, выполните следующие указания:

1. Создайте абстрактный класс **Vehicle** со следующими методами:
   - `virtual void move() const = 0;`
   - `virtual std::string type() const = 0;`
   - виртуальный деструктор.
2. Создайте следующие производные классы:
   - Car: выводит "Car is driving on the road."
   - Boat: выводит "Boat is sailing on water."
   - Plane: выводит "Plane is flying in the sky."
   - Метод `type()` производных классов должен выводить строку – название соответствующего класса, например `return "Car"`
3. В main() создайте вектор указателей на **Vehicle** и наполните его объектами разных типов.
4. Выведите информацию о типе и поведении каждого объекта в цикле:

```
for (const auto* v : vehicles) {
    std::cout << v->type() << ": ";
    v->move();
}
```

Дополнительно (по желанию):

- Добавьте класс AmphibiousVehicle, сочетающий Car и Boat.
- Реализуйте счётчик всех созданных транспортных средств (через static поле).
- Добавьте простую сериализацию в текстовый файл: <тип>;<поведение>.

В C++ существует механизм, позволяющий определить динамический тип объекта во время выполнения — это **RTTI (Run-Time Type Information)**.

Если у класса есть хотя бы одна виртуальная функция (например, виртуальный деструктор), то можно использовать:

```
#include <typeinfo>

Vehicle* v = new Car();
std::cout << typeid(*v).name() << std::endl;
```

Вывод зависит от компилятора, например:

- GCC/Clang: "3Car" (mangled name)
- MSVC: "class Car"

Попробуйте этот механизм реализовать в задании.

Задание 2. Полиморфизм в системе классов учебного центра

Требуется реализовать полиморфный вызов методов производных классов системы, описанной в упражнениях предыдущего практического занятия.

Практическое занятие 14. Использование шаблонных функций и классов

Упражнение 1. Создание шаблонной функции сортировки массива

В этом упражнении вы создадите шаблонную функцию сортировки массива элементов различных типов.

1. Создайте новое приложение и добавьте файл исходного кода.
2. В файле реализуйте функцию сортировки, принимающую два параметра: массив целых чисел и его размер:

```
void sorting (int arr[], int size){
      int j = 0;
      for (int i = 0; i < size; i++) {
            int x = arr[i];
            for (j = i - 1; j >= 0 && x < arr[j]; j--)
                  arr[j+1] = arr[j];
         arr[j + 1] = x;
      }
}
```

3. В методе `main()` создайте массив, вызовите функцию сортировки и отобразите отсортированный массив на экране:

```
int main()
{
  int arr[] = {9,3,17,6,5,4,31,2,12};
  int k1 = sizeof(arr)/sizeof(arr[0]);
  sorting(arr, k1);
  for (int i = 0; i < k1; i++ ) cout << arr[i] << ";";
}
```

4. Постройте и протестируйте приложение.
5. На основе функции сортировки создайте шаблон функции:

```
template<class T>
void sorting (T arr[], int size){
      int j = 0;
      for (int i = 0; i < size; i++ ){
            T x = arr[i];
            for (j = i - 1; j >= 0 && x < arr[j]; j--)
                arr[j+1] = arr[j];
            arr[j + 1] = x;
      }
```

```
}
```

6. В методе `main()` создайте два массива, один с вещественным, другой с символьным типами данных. Протестируйте работу шаблонной функции:

```
int arr[] = {9,3,17,6,5,4,31,2,12};
double arrd[] = {2.1, 2.3,1.7,6.6,5.3,2.44,3.1,2.4,1.2};
char arrc[] = "Hello, word";
int k1 = sizeof(arr)/sizeof(arr[0]);
int k2 = sizeof(arrd)/sizeof(arrd[0]);
int k3 = sizeof(arrc)/sizeof(arrc[0]) - 1;
sorting(arr, k1);
for ( int i = 0; i < k1; i++ ) cout << arr[i] << ";";
cout << endl;
sorting (arrd, k2);
for ( int i = 0; i < k2; i++ ) cout << arrd[i] << ";";
cout << endl;
sorting (arrc, k3);
for ( int i = 0; i < k3; i++ ) cout << arrc[i] << ";";
cout << endl;
```

7. Создайте еще одну шаблонную функцию для вывода массивов различных типов на экран.
8. Постройте и протестируйте приложение.

Упражнение 2. Создание шаблонной функции для работы с кортежем

В этом упражнении вы реализуете функцию-шаблон для вывода на экран содержимое кортежа из трех значений любого типа, а не так как это было сделано в упражнении 4 практического занятия 7 (функция printTupleOfThree принимала кортеж строго определенного типа набор элементов кортежа).

1. Откройте решение упражнения 4 практического занятия 7.
2. Добавьте перед определением функции определение шаблона и измените тип кортежа со списком параметров на шаблонный тип:

```
template<class Tuple>
void printTupleOfThree(Tuple t)
{...
```

3. Создайте новую переменную кортежа и проверьте работу шаблонной функции, например:

```
auto t3 = std::make_tuple(v1[0], v1[1], v2[2]);
printTupleOfThree(t3);
```

Обратите внимание, что теперь можно создавать кортеж из элементов любого типа, а не как раньше – <string, int, double> . Но ограничение на количество элементов осталось – по-прежнему функция выводить только три элемента кортежа. Это ограничение будет снято в следующем упражнении.

Упражнение 3. Использование шаблонной функции для работы с кортежем любого размера

В этом упражнении для решения задачи универсализации функции печати вы примените вариативные шаблоны, способные принимать переменное количество аргументов.

1. Создайте новый проект.
2. Создайте два варианта объявления шаблонного класса `TuplePrinter`, в котором определите одну статическую функцию с именем `print`. Эта реализация называется рекурсивным экземпляром шаблона:

```
#include <iostream>
#include <tuple>
#include <vector>
using namespace std;

template<class Tuple, std::size_t N>
struct TuplePrinter {
    static void print(const Tuple& t)
    {
        TuplePrinter<Tuple, N - 1>::print(t);
        cout << ", " << get<N - 1>(t);
    }
};

template<class Tuple>
struct TuplePrinter<Tuple, 1> {
    static void print(const Tuple& t)
    {
        cout << get<0>(t);
    }
};
```

3. Объявите функцию `printTuple()` как функцию вариативного шаблона, которая вызывает функцию-член `print()` класса `TuplePrinter`:

```
template<class... Args>
void printTuple(const tuple<Args...>& t)
{
    cout << "(";
    TuplePrinter<decltype(t), sizeof...(Args)>::print(t);
    cout << ")" << endl;
}
```

4. В функции `main()` добавьте векторы для исходных данных кортежей из прошлого упражнения и создайте два кортежа с элементами разных

типов и различного количества, затем вызовите созданную функцию `printTuple()` для каждого кортежа:

```
int main()
{
    vector<string> v1{ "one", "two", "three", "four", "five", "six" };
    vector<int> v2 = { 1, 2, 3, 4, 5, 6 };
    vector<float> v3 = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };

    auto t1 = std::make_tuple(v1[0], v1[1], v3[0]);
    auto t2 = std::make_tuple(v1[0], v1[1], v2[1], v3[0], v3[1]);
    printTuple(t1);
    printTuple(t2);

    return 0;
}
```

5. Проверьте, что в результате вы получили функцию, совместимую с объектами кортежей разного размера и способную выводить все их члены в поток cout.

Упражнение 4. Использование шаблонного класса массива

В этом упражнении вы на основе класса массива, в котором есть методы для вычисления суммы и среднего значения, хранимых в массиве чисел типа int создадите шаблонный класс.

Откройте файл исходного кода программы templ_class_mas.cpp (папка Source_template)

Изучите код. Программа распределяет 100 элементов массива, а затем заносит в массив 50 значений с помощью метода `add_value`. В классе `array` переменная `index` отслеживает количество элементов, хранимых в данный момент в массиве. Если пользователь пытается добавить больше элементов, чем может вместить массив, функция `add_value` возвращает ошибку. Как видите, функция `average_value` использует переменную `index` для определения среднего значения массива. Программа запрашивает память для массива, используя оператор `new`.

1. Создайте новое приложение и добавьте пустой файл исходного кода.
2. На основе класса массива целых чисел создайте шаблон класса, который создает общий класс array:

```
template<class T, class T1> class array
{
public:
   array(int size);
   T1 sum();
   T average_value();
   void show_array();
```

```
        int add_value(T);
    private:
        T *data;
        int size;
        int index;
    };
```

3. Перед каждой функцией класса укажите запись со словом `template`. Кроме того, сразу же после имени класса укажите типы класса, например array<T, T1>::average_value:

```
template<class T, class T1> T array<T, T1>::average_value()
{
    T1 sum = 0;
    for (int i = 0; i < index; i++) sum += data[i];
    return (sum / index);
}
```

4. Измените остальные методы.
5. В результате методы класса примут следующий вид:

```
template<class T, class T1> array<T, T1>::array(int size)
{
   data = new T[size];
   if (data == NULL)
   {
      cerr<< "Error memory ---- exit program" <<endl;
      exit(1);
   }
   array::size = size;
   array::index = 0;
}
template<class T, class T1> T1 array<T, T1>::sum()
{
   T1 sum = 0;
   for (int i = 0; i < index; i++) sum += data[i];
   return(sum);
}

template<class T, class T1> T array<T, T1>::average_value()
{
   T1 sum = 0;
   for (int i = 0; i < index; i++) sum += data[i];
   return (sum / index);
}

template<class T, class T1> void array<T, T1>::show_array()
```

```
{
   for (int i = 0; i < index; i++) cout << data[i] << ' ';
   cout << endl;
}

template<class T, class T1> int array<T, T1>::add_value(T value)
{
   if (index == size)
   return(-1);
   else
   {
      data[index] = value;
      index++;
      return(0);
   }
}
```

6. В методе `main()` создайте два массива:

```
array<int, long> numbers(100);
array<float, float> values(200);
```

7. С помощью метода `add_value` реализуйте заполнение массивов, а затем вычисление суммы и среднего значения:

```
int i;
for (i = 0; i < 50; i++) numbers.add_value(i);
numbers.show_array();
cout << "Sum =  " << numbers.sum () << endl;
cout << "Average =  " << numbers.average_value() << endl;

for (i = 0; i < 100; i++) values.add_value(i * 100);
values.show_array();
cout << "Sum = " << values.sum() << endl;
cout << "Average =  " << values.average_value() << endl;
```

8. Постройте и протестируйте приложение.

Упражнение 5. Шаблонная функция с ограничением типов

В этом упражнении вы реализуете возможность при применении шаблонной функции ограничить типы, т.е. не допустить те типы, которые не предназначены для шаблонной функции. Для решения данной задачи в версии C++20 введены концепты (**concepts)** — способ ограничить шаблонные параметры.

1. Создайте новое приложение и добавьте файл исходного кода.
2. В файле реализуйте функцию **Factorial**, доступную только для целочисленных типов (для ограничение типов используйте std::integral):

```
#include <concepts>
```

```
template <std::integral T>
T Factorial(T n) {
    if (n <= 1) return 1;
    return n * Factorial(n - 1);
}
```

3. В методе `main()` создайте две переменные – целого и вещественного типов, вызовите функцию факториала для них, проверьте поведение функции при передаче вещественного типа.

Контрольные задания.

Задание 1. Шаблонная функция обработки массива

Напишите шаблон функции, возвращающей среднее арифметическое всех элементов массива.

Аргументами функции должны быть имя и размер массива (типа int).

В функции `main()` проверьте работу с массивами типа int, long, double и char.

Задание 2. Шаблонная функция вывода данных в поток

Требуется реализовать шаблонную функцию `Print`, которая выводит в поток `std::cout` элементы переданного контейнера через указанную качестве параметра строку-разделитель.

Первый аргумент функции — контейнер. Требуется гарантировать, что по этому контейнеру можно проитерироваться с помощью стандартного цикла range-based for, и что элементы контейнера можно вывести в поток `std::cout` с помощью стандартного оператора <<.

Второй аргумент функции — строка-разделитель, которую надо печатать между элементами. В конце необходим перевод строки \n.

Пример вызова:

```
int main() {
      std::vector<int> data = {1, 2, 3};
      Print(data, ", ");          // на экране: 1, 2, 3
}
```

Рекомендация:

Используйте константные ссылки для получения параметров при итерации в цикле, чтобы избежать лишних копирований.

Задание 3. Шаблонная функция maxOfTwo с ограничением

Напишите шаблонную функцию `maxOfTwo` с ограничением, который требует наличие оператора < (используйте концепт **Comparable**). Функция должна возвращать максимум из двух значений. Проверьте работу функции для типов int, double и пользовательского класса с оператором <.

Задание 4. Шаблонная функция для сортировки (опционально)

Это задание выполняется для претендующих на оценку «отлично».

Требуется:

1. Создать собственный concept **Sortable**, который требует наличие операций сравнения < и >.
2. Написать шаблон функции `is_sorted`, которая проверяет, отсортирован ли контейнер (используя итераторы) для типов, удовлетворяющих **Sortable**.
3. Проверить работу функции на std::vector<int>, std::vector<std::string>, а также попытаться использовать с типом, не поддерживающим сравнение (например, std::vector<std::complex<double>>).

Практическое занятие 15. Использование STL

Упражнение 1. Создание списка студентов

В этом упражнении вы для представления студентов используете последовательный контейнер – список (**list<T>**). Выбор списка в качестве контейнера объясняется тем, что он обеспечивает константное время вставки и удаления в любом месте последовательности.

1. Создайте проект и добавьте файлы исходного проекта из папки Lab_STL.
2. Постройте решение и протестируйте его работу.

Вы должны ввести данные о студенте. Изучите содержание файлов. Обратите внимание на связь между классами **Student** и **IdCard**. Класс **Group** представляет собой учебную группу. В этом упражнении Вы реализуете связь между классами **Group** и **Student**, используя контейнеры STL.

3. В файл Group.h добавьте следующие директивы для работы с STL:

```
#include <list>
#include <algorithm>
```

4. В закрытую часть класса **Group** добавьте список студентов – контейнер **list** и итератор для работы с ним:

```
list <Student> masSt;
list <Student>::iterator iter;
```

5. В открытой части класса **Group** объявите прототипы методов для:

расчета размера контейнера

```
int getSize();
```

добавления студента в контейнер и удаление его оттуда

```
void addStudent(Student newStudent);
void delStudent(Student oldStudent);
```

поиска студента по фамилии и имени

```
Student findStudent(string, string);
```

сортировки списка

```
void GroupSort();
```

вывода информации о содержании контейнера

```
void GroupOut();
```

6. Реализуйте в файле Group.cpp эти методы:

```
int  Group::getSize()
{
      return masSt.size();
}


void Group::addStudent(Student newStudent)
{
      masSt.push_back(newStudent);
}


void Group::delStudent(Student oldStudent)
{
      masSt.remove(oldStudent);
}


void Group::GroupOut()
{
      iter = masSt.begin();
      while(iter != masSt.end() )
            (*iter++)->display();
}


Student Group::findStudent(string searchName, string
searchLastName)
{
      Student temp(searchName, searchLastName);
      iter = find(masSt.begin(), masSt.end(), temp);
      return(*iter);
}
```

7. В классе **Student** объявите четыре дружественные операторные функции, которые будут перегружать операторы сравнения, применяемые к объектам-студентам:

```
friend bool operator< (const Student&, const Student&);
friend bool operator> (const Student&, const Student&);
friend bool operator== (const Student&, const Student&);
friend bool operator!= (const Student&, const Student&);
```

8. Реализуйте в файле Student.cpp эти операторные функции:

```
bool operator== (const Student& p1, const Student& p2)
{
return (p1.name == p2.name && p1.last_name == p2.last_name ) ?
true :false;
}

bool operator< (const Student& p1, const  Student& p2)
{
      if(p1.last_name == p2.last_name)
            return (p1.name < p2.name) ? true :false;
      return (p1.last_name < p2.last_name ) ? true :false;
}

bool operator!= (Student& p1,  Student& p2)
{ return !(p1 == p2); }

bool operator> (Student& p1,  Student& p2)
{ return !(p1 < p2) && !(p2 == p2); }
```

Обратите внимание на перегрузку оператора<, он задает метод сортировки элементов контейнера. В данном случае он определен таким образом, чтобы сортировались фамилии, а в случае их совпадения и имена.

9. В методе `main()` после создания студента `student02` создайте нескольких новых студентов, например

```
Student student03 ("Петр", "Петров", idc2);
Student student04 ("Семен", "Смирнов", idc);
Student student05 ("Саша", "Коен", idc2);
Student student06 ("Дмитрий", "Ионов", idc);
```

10. Для тестирования работы класса **Group** создайте группу, например 1557:

```
Group gr1957("1957");
```

11. Добавьте созданных ранее студентов в группу:

```
gr1957.addStudent(student02);
gr1957.addStudent(student03);
gr1957.addStudent(student04);
gr1957.addStudent(student05);
gr1957.addStudent(student06);
```

Такой способ создания группы студентов применен для простоты, на практике ввод данных лучше реализовать в цикле.

12. Определите размер группы:

```
int k = gr1957.getSize();
```

13. Вызовите функцию сортировки списка:

```
gr1957.GroupSort();
```

14. Реализуйте вывод данных о группе:

```
cout << "В группе "<< gr1957.getName() <<" "<<k<<" ст." << endl;
        gr1957.GroupOut();
```

15. Постройте и протестируйте приложение. Проверьте порядок вывода студентов.
16. Проверьте удаление конкретного студента из контейнера. Для этого добавьте следующий код (можно сразу после кода добавления студентов в контейнер):

```
gr1957.delStudent(student04);
```

17. Постройте и протестируйте приложение.
18. Удалите код `gr1957.delStudent(student04);`
19. С помощью функции поиска найдите нужного студента и удалите его из списка группы:

```
gr1957.delStudent(gr1957.findStudent("Семен", "Смирнов"));
```

20. Повторите вывод списка группы, проверьте, что удаленный студент отсутствует в списке.
21. Постройте и протестируйте приложение.

Упражнение 2. Организация студентов с помощью мультимножества

В этом упражнении вы для представления студентов используете ассоциативный контейнер – мультимножество. Элементами этого контейнера будут указатели на объекты класса **Student**. Хранение указателей вместо объектов является хорошим решением, особенно в случае больших объемов информации. Такой подход становится эффективным за счет избегания копирования каждого объекта при помещении его в контейнер. Тем не менее, возникает проблема сортировки, поскольку объекты будут выстроены по адресам указателей, а не по каким-то собственным атрибутам. Для решения этой проблемы необходимо создать отдельный функциональный объект, задающий метод сортировки.

В этом упражнении вы создадите мультимножество для хранения указателей на объекты класса **Student**, определите функциональный объекта **compareStudent**, для того чтобы сортировка производилась автоматически по требуемым атрибутам.

Для учебных целей вы переделаете приложение из прошлого упражнения.

1. В файл Student.h добавьте функциональный объект для сравнивания содержимого указателей на объекты:

```
class compareStudent
{
public:
      bool operator() (const Student* ptrSt1, const Student*
ptrSt2) const
      {return *ptrSt1 < *ptrSt2; }
};
```

Функция **operator**() имеет два аргумента, являющихся указателями на персональные данные, и сравнивает значения их содержимого, а не просто значения указателей.

2. В файл Group.h добавьте директиву для работы с множеством:

```
#include <set>
```

3. Замените в классе **Group** список студентов – контейнер **list** и итератор на контейнер **multiset** и соответствующий итератор (при определении контейнера определите созданный функциональный объект):

```
multiset<Student*, compareStudent> masSt;
multiset<Student*, compareStudent>::iterator iter;
```

4. В классе **Group** замените методы контейнера list на соответствующие методы множества (измените и прототипы этих методов):

```
void Group::addStudent(Student* newStudent)
{
      masSt.insert(newStudent);
}
void Group::delStudent(Student* oldStudent)
{
      masSt.erase(oldStudent);
}
```

5. В методе вывода информации о содержании контейнера измените вызов метода:

```
void Group::GroupOut()
{
      iter = masSt.begin();
      while( iter != masSt.end() )
            (*iter++)->display();
}
```

6. В методе поиска студента внесите следующие изменения (измените и прототип этого метода):

```
Student* Group::findStudent(string searchName, string
searchLastName)
{
      Student *temp = new Student(searchName, searchLastName);
      iter = masSt.lower_bound(temp);
      delete temp;
      return (*iter);
}
```

Метод `lower_boimd()` получает в качестве аргумента искомое значение того же типа и возвращает итератор, указывающий на первую запись множества, значение которой не меньше аргумента (что значит «не меньше», в каждом конкретном

случае определяется конкретным функциональным объектом, используемым при определении множества).

Если требуется реализовать поиск на интервале, то потребуется также функция `upper_bound()` , которая возвращает итератор, указывающий на элемент, значение которого больше, чем аргумент. Обе эти функции позволяют задавать диапазон значений в контейнере.

7. Метод для расчета размера контейнера оставьте без изменений.
8. Удалите метод сортировки.
9. В методе `main()` измените код создания объекта `student02` и в соответствии с этим вызов методов:

```
Student* student02 = new Student(name, last_name, idc);
```

10. Измените код создания объектов – студентов:

```
Student* student03 = new Student("Петр", "Петров", idc2);
Student* student04 = new Student("Семен", "Смирнов", idc);
Student* student05 = new Student("Саша", "Коен", idc2);
Student* student06 = new Student("Дмитрий", "Ионов", idc);
```

11. Код создания группы, добавления студентов, поиска и удаления требуемого студента не изменяйте.
12. Постройте и протестируйте приложение. Проверьте, что список группы отображается сразу в отсортированном виде.

Контрольные задания

Задание 1. Организация хранения данных в контейнере

Требуется написать программу, которая позволит присваивать оценки студентам, указывая только имя студента.

Для организации данных используйте контейнер **map** (контейнер map – это класс, в котором все элементы хранятся в виде пары ключ-значение), ключ должен быть уникальным и использоваться для доступа к связанной паре: имя студента – ключ, оценка (тип char) - значение.

Указание. Создайте структуру (класс в принципе не требуется для этой задачи) StudentGrade с двумя элементами: имя студента (std::string) и оценка (char).

Задание 2. Поиск максимального чётного числа в векторе

Напишите функцию **FindMaxEven**, которая возвращает наибольшее чётное число в векторе std::vector<int>, причем если чётных чисел нет, то следует выбросить исключение std::runtime_error.

Рекомендация: используйте алгоритмы с лямбда-функцией, итераторы STL.