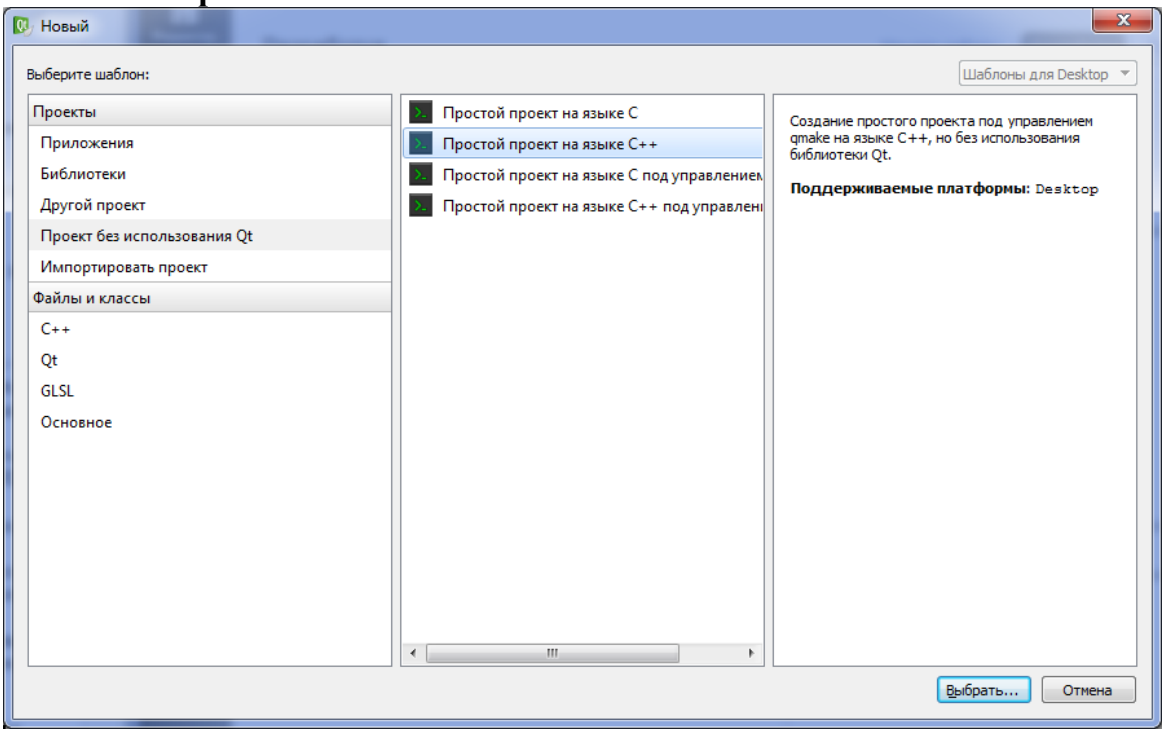
СОДЕРЖАНИЕ

Практическое занятие 1. Создание программы на языке C++ при помощи Qt Creator

Упражнение 1. Использование интегрированной среды разработки для создания программы

В этом упражнении dы приобретете навыки создания проекта для разработки программы на языке C++.

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



4. В поле **Название** введите имя проекта – **MyFistProg.**
5. 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**.
6. Нажмите кнопку **Далее** и в окне **Управление проектом** нажмите **Завершить.**
7. Откроется окно проекта.
8. Изучите структуру проекта и содержимое файла main.cpp:

```
#include <iostream>

using namespace std;

int main()
```

```
{
    cout << "Hello World!" << endl;
    return 0;
}
```

9. С помощью меню **Сборка** запустите приложение на выполнение.
10. Введите текст программы:

```
#include <iostream>
using namespace std;
//  Программа деления двух чисел
int main()
{
    double x; /* объявления переменных*/
    double a, b;
    cout << "\nВведите a и b:\n";   /* вывод приглашения */
    cin >> a;    /* ввод с клавиатуры значения a */
    cin >> b;    /* ввод с клавиатуры значения b */
    x = a / b;      /* вычисление значения x */
    cout << "\nx = " << x << endl;  /* вывод результата на экран */
    return 0;
}
```

11. С помощью меню **Сборка** постройте проект, исправьте ошибки и запустите приложение на выполнение**.**

Упражнение 2. Преобразование типов данных

В этом упражнении Вы исследуете преобразование типов данных.

1. В программе из прошлого упражнения замените объявление переменной `x`: вместо `double` укажите тип `int`.
2. Запустите программу.
3. 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.

4. Верните для переменной `x` правильный тип – `double` и постройте приложение.

Упражнение 3. 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 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 **math.h** и требуется его указать в директиве:

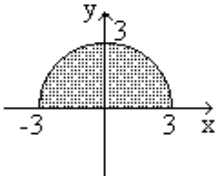
```
#include<math.h>
```

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. Определение високосного года

Дано натуральное число.

Требуется определить, является ли год с данным номером високосным. Если год является високосным, то выведите YES, иначе выведите NO.

- Год является високосным, если его номер кратен 4, но не кратен 100, а также, если он кратен 400.

Задание 3. 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.
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`.
6. Постройте и протестируйте приложение.

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.

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:

```
     temp = a;
     while (temp!=b)
     {
          a = temp;
          if (a<b)
          {
               temp = a;
               a = b;
```

```
            b = temp;
        }
        temp = a - b;
        a = b;
    }
cout << "НОД = " << b << 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. 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 можно воспользоваться циклом `for` и оператором перехода `continue`:

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

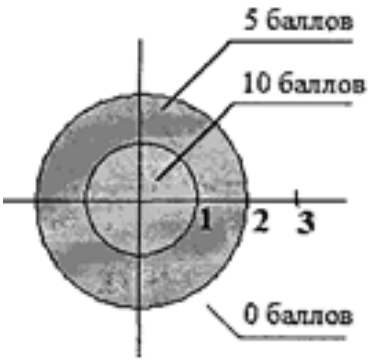
Упражнение 4. Стрельба по мишени

В этом упражнении требуется разработать программу, имитирующую стрельбу по мишени.

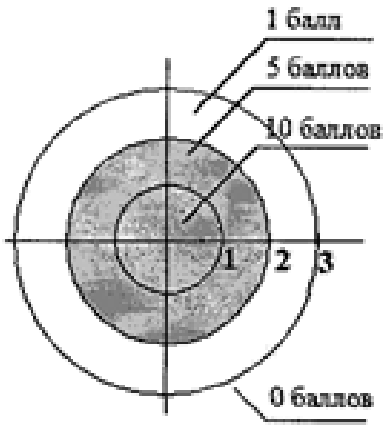
Пользователь вводит данные о выстреле в виде пары чисел – координат `x` и `y` заранее известное количество раз.

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

Вариант мишени выберите самостоятельно.



Вариант 1.



Вариант 2.

Дополнительные задания *(выполнять не обязательно)*:

- 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.

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.

1. Создайте новый проект и добавьте файл исходного кода.
2. В функции `main()` объявите строковую переменную и запросите имя с клавиатуры (обратите внимание на использование строкового типа):

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

int main ()
{
     SetConsoleOutputCP(1251);
     SetConsoleCP(1251);
string name;
cout << "Введите свое имя" << endl;
cin >> name;
cou t<< name << ", " << "здравствуйте!" << 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 << ", " << "здравствуйте!" << endl;
}
```

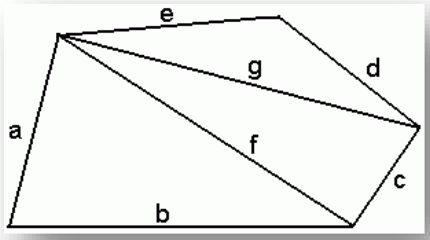
4. В функции вместо непосредственного вывода на экран вызовите новую функцию с передачей ей в качестве параметра введенное имя:

```
privet(name);
```

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

Упражнение 2. Расчет площади сложной фигуры

Требуется написать программу для вычисления площади выпуклого пятиугольника.



Исходные данные: координаты вершин пятиугольника (значения задайте самостоятельно).

Для решения задачи следует использовать метод декомпозиции: сначала получить площадь треугольников, а затем их сложить.

Для расчета сторон и площади треугольника использовать отдельные соответствующие функции. Длину каждой стороны можно определить по следующей формуле:

$$a = \sqrt{(x_b - x_c)^2 + (y_b - y_c)^2},$$

где x_b, y_b и x_c, y_c координаты двух точек отрезка.

Площать треугольника по трем сторонам определялась в упражнении 4 практического занятия 1.

Упражнение 3. 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 или их типом.

1. В проекте упражнения 1 добавьте вторую функцию `privet`, но с двумя параметрами:

```
void privet(string name, int k)
{
  cout << name << " " << ", еще раз" << "здравствуйте! "
  << "Вы ввели " << k << endl;
}
```

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

```
int main ()
{
  ...
  int k;
  ...
 cout << "Введите Ваше любимое число" << 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 4. Расчет площади равностороннего треугольника

В этом упражнении добавьте в приложение рассчитывающее площадь треугольника (занятие 1, упражнение 4) перегруженную функцию, которая будет принимать один параметр – сторону и вычислять площадь равностороннего треугольника.

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 или указатель на нее. В этом случае 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. Вычисление корней квадратного уравнения

Требуется реализовать функцию вычисления 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 может выглядеть следующим образом:

```
int Myroot(double a, double b, double c, double &x1, double &x2);
```

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 и среднего значения на экран.
7. Постройте и протестируйте приложение.

8. Добавьте в программу новые возможности обработки данных массива:
    a. расчет суммы отрицательных элементов,
    b. расчет суммы положительных элементов,
    c. расчет суммы элементов с нечетными номерами,
    d. расчет суммы элементов с четными номерами.
9. Дополнительные задания:
    a. найти максимальный и минимальный элементы массива и вывести их индексы,
    b. рассчитать произведение элементов массива, расположенных между максимальным и минимальным элементами
10. Постройте и протестируйте приложение.

Упражнение 2. 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 3. 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.

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.

Упражнение 2. Запись текста в файл

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

Упражнение 3.Сохранение данных в текстовый файл

В решении упражнения 3 занятия 5 создайте текстовый файл и запишите в него два массива: исходный и отсортированный.

Практическое занятие 7. Применение структур

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 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 методов сложения и вывода на экран константные ссылки, которые запрещают изменение объекта внутри функции:

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

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.

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;
}

// Установкапромежуточныхоценок
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 3. 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, stringlast_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 4. Сохранение данных в файл

В этом упражнении Вы реализуете возможность сохранения данных после окончания работы с программой: запись данных в текстовый файл.

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:

```
studentneo->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.

```
// 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.

Практическое занятие 9. Обработка исключительных операций

Упражнение 1. 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. Создайте приложение с файлом исходного кода Student01_main.cpp (файл находится в папке Labs или 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.

Упражнение 2. 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. упражнение 4 практического занятия 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.

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 7.
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. Реализация класса Triangle

В этом упражнении 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));
}
```

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+ (Distance) const;
};
```

3. Определите операторную функцию сложения двух расстояний:

```
Distance Distance::operator+ (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 перегру операцию вычитания, 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).

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.

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 `stringstream` (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.

Упражнение 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. 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.

Упражнение 3. Использование шаблонного класса

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.

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.

Литература

1. Глушаков С.В., Дуравкина Т.В. Программирование на С++. – М.: АСТ, 2009. – 685 с.
2. Джамса К. Учимся программировать на языке С++. — М.: Мир, 2009.- 320 с.
3. Лафоре Р. Объектно-ориентированное программирование в С++. - СПб.:Питер, 2012. – 922 с.
4. Либерти Д. Освой самостоятельно С++ за 21 день. – М.: Диалектика-Вильямс, 2010. – 850с.
5. Шилдт Г. Искусство программирования на С++. – СПб.: БХВ-Петербург, 2011. – 496 с.
6. Стивен Прата. Язык программирования С++. Лекции и упражнения, 6-е изд. : Пер. с англ. – М. : ООО "И.Д. Вильямс", 2012. - 1248 с.