Федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

**Отчет**

**Тема: Творческое задание**

Вариант 17

Выполнил работу

студент группы РИС-19-1б

Мазитов Т.Э.

Проверила

доцент кафедры ИТАС

Полякова О.А.

Работу выполнил:

Пермь 2020

**Содержание**

Введение 3

Используемые инструменты 4

Решение задачи Коммивояжера 5

Разработка калькулятора 12

Заключение 18

**Введение**

Заключительным этапом курса “Информатики” и “Основы алгоритмизации и программирования” во втором семестре является творческая работа. Первая ее часть ставит задачу разработки программы, решающей проблему коммивояжера с использованием метода ветвей и границ и визуализацией графа с помощью графических библиотек. Вторая часть предлагает студентам разработать интерфейс и алгоритм работы калькулятора для работы с различными входными данными.

Творческая работа подразумевают использование накопленных за курс знаний и демонстрацию таланта студентов.

**Используемые инструменты**

Интегрированная среда разработки:

* Microsoft Visual Studio 2019;

Работа с видеорядом:

* Запись экрана компьютера – Bandisoft Bandicam;
* Редактирование видео – Bandisoft Bandicut;

Графический интерфейс программ:

* GLUT – открытая библиотека утилит для приложений под OpenGL;
* Windows Forms — интерфейс программирования приложений, отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework;

Язык программирования:

* C++ – компилируемый, статически типизированный язык программирования общего назначения, поддерживающий парадигму объектно-ориентированного программирования.
* C++/CLI — надстройка над C++, язык для среды программирования Microsoft .NET;

**Решение задачи Коммивояжера**

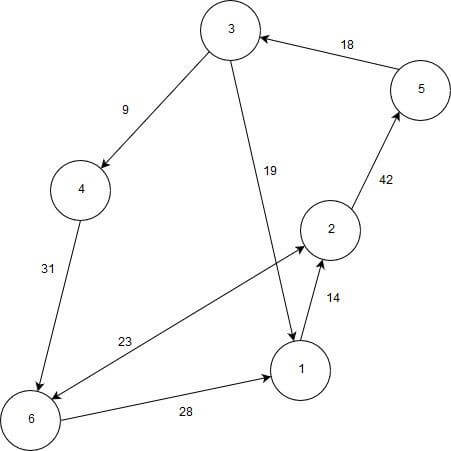
**Цели работы**

* разработать программу для решения задачи коммивояжера методом ветвей и границ:
* реализовать графическую часть проекта с помощью openGl;
* разработать программу, которая будет универсальной на любом наборе исходных данных (в качестве исходного взять граф из своего варианта).

**Ход работы**

Задача коммивояжёра (коммивояжёр — бродячий торговец) заключается в отыскании самого выгодного маршрута, проходящего через указанные узлы хотя бы по одному разу. Как правило, указывается, что маршрут должен проходить через каждый город только один раз – в таком случае выбор осуществляется среди гамильтоновых циклов.

Простейшие методы решения задачи коммивояжёра: полный лексический перебор, жадные алгоритмы, метод минимального островного дерева. На практике применяются различные модификации более эффективных методов: метод ветвей и границ и метод генетических алгоритмов, а также алгоритм муравьиной колонии.

****

*Граф для варианта 17*

**Метод ветвей и границ**

Последовательность шагов в методе ветвей и границ:

1. составить исходную смежную матрицу: если из узла нет пути в другой узел, то в соответствующем элементе матрицы ставится “-1”;
2. найти минимальные значения в каждой строке и вычесть их из каждого элемента строки. Сумму минимальных значений записать;
3. найти минимальные значения в столбцах и вычесть их из каждого элемента столбца. Сумму минимальных значений записать;
4. текущая длина пути - сумма двух записанных чисел;
5. посчитать штрафы у нулей в матрице – сумма минимального положительного значения по строке и минимального положительного значения по столбцу;
6. выбрать ноль с максимальным значением штрафа;
7. составить матрицу, где на место выбранного нуля поставить “-1”. В элементе матрицы, отображающей обратный переход поставить “-1”;
8. составить матрицу, где на места всех элементов строки выбранного нуля поставить “-1”, и где на места всех элементов столбца выбранного нуля поставить “-1”;
9. для каждой полученной матрицы произвести шаги 2-4;
10. матрица, с минимальным значением пути становится исходной. Если выбрана первая, включающая выбранный ноль, то его номер строки и столбца записываются в ответ как переход из узла с номером строки в узел с номером столбца. Иначе, данных переход не входит в путь решения задачи;
11. шаги 5 -10 повторяются пока, все элементы матрицы не станут равны “-1”;
12. значение длины пути конечной матрицы – ответ к задаче Коммивояжера.

**Функции и методы классов**

Основной код программы реализован в методах класс *Graph*. Ключевыми в решении задача являются методы *Commute\_problem, hamilton, zero\_mark, COM\_rows\_columns.*

Метод *hamilton* принимает в качестве аргумента начальный узел и проверяет для него наличие пути Гамильтона (содержащий каждую вершину графа ровно один раз). Если такой путь найден, то метод возвращает значение *true* и можно продолжать решить задачу для текущего графа. Иначе возвращается значение *false* и решение задачи для этого графа невозможно.

Метод *zero\_mark* считает количество штрафов у нуля в смежной матрице графа и возвращает найденное значение. В качестве аргумента принимает два целых числа: номер строки и столбца в матрице смежности, где находится рассматриваемое число ноль.

Метод *COM\_rows\_columns* принимает смежную матрицу в качестве аргумента работает в три этапа:

1. находит минимальное значение в каждой строке, сохраняет их сумму и вычитает из каждой строки соответствующее минимальное значение;
2. находит минимальное значение в каждом столбце, сохраняет их сумму и вычитает из каждого столбца соответствующее минимальное значение;
3. складывает полученные суммы в значение и возвращает его.

В методе *Commute\_problem* описан алгоритм решения задачи с помощью метода ветвей и границ. Суть подхода: нужно разделить огромное число перебираемых вариантов на классы и получить оценки (снизу – в задаче минимизации, сверху – в задаче максимизации) для этих классов, чтобы иметь возможность отбрасывать варианты не по одному, а целыми классами. Трудность состоит в том, чтобы найти такое разделение на классы (ветви) и такие оценки (границы), чтобы процедура была эффективной.

Графической часть программы реализована в двух методах класса:

* *DrawGraph* – основной метод, который нужно вызвать для начала отрисовки графа. Метод вызывает функции, описанные ниже, не принимает и не возвращает значений;
* *drawVertex* – рисует все узлы графа в форме кругов и текст их номеров.

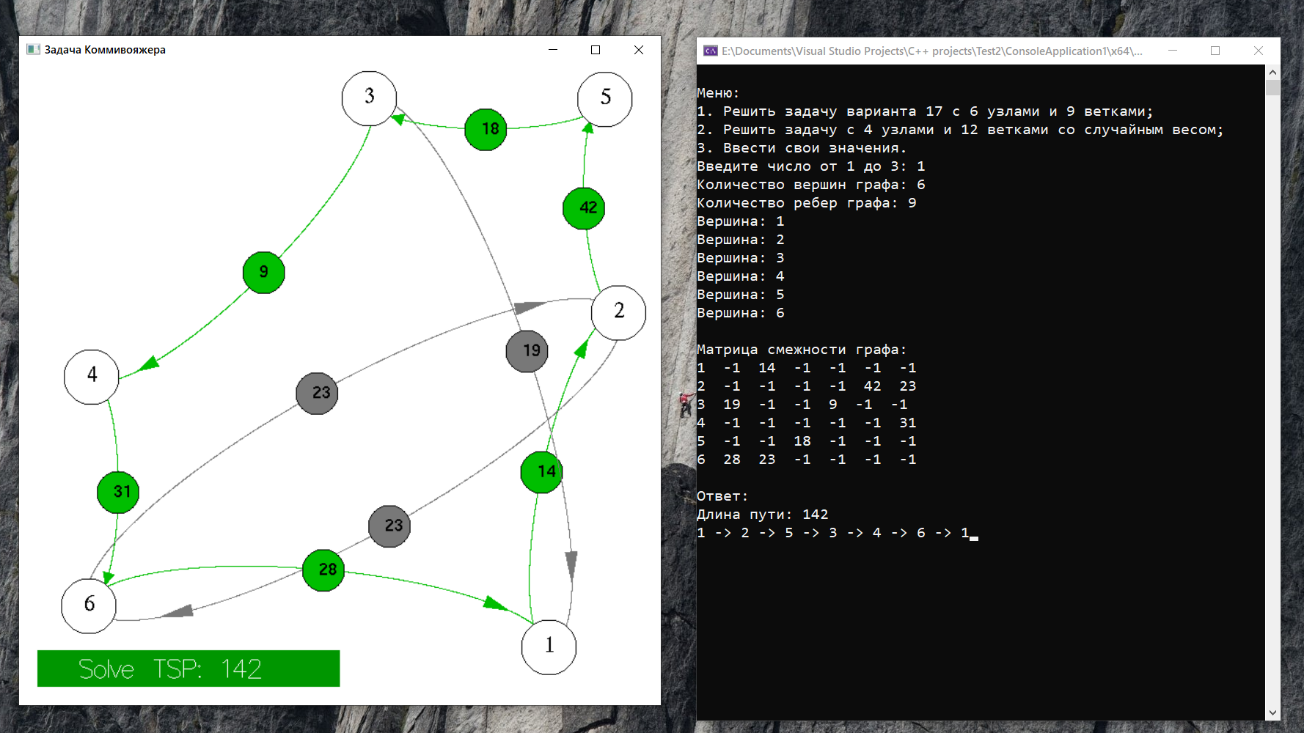
а также 6 функциях:

* *setCord* – принимает количество всех узлов и номер текущего узла, для которого вычисляет координаты и сохраняет их в массив структур *VectC*[];
* *drawCircle* – рисует круг определенного цвета в заданных координатах;
* *drawText* – рисует текст определенного цвета в заданных координатах. Если текст является весом ребра, то предварительно в координатах рисуется круг.
* *drawLine* – рисует линию между двумя точками;
* *drawTR* – рисует равнобедренный острый треугольник по двум точкам;
* *drawHalfEllipse* – рисует половину эллипса по двум точкам.

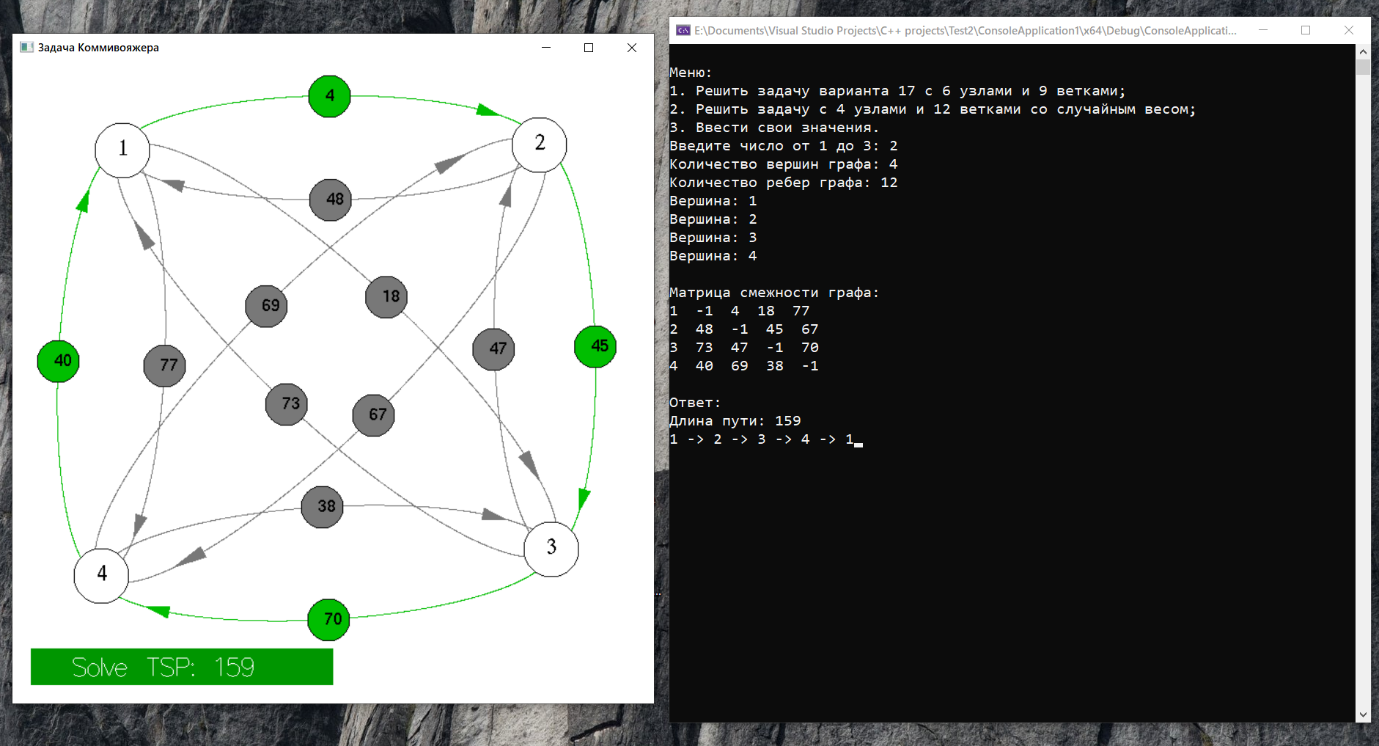
Взаимодействие мыши с графом реализовано через метод move:

* *move* – принимает координаты мыши и проверяет случаи наложения. Есть таких нет, то узлу присваиваются новые координаты.

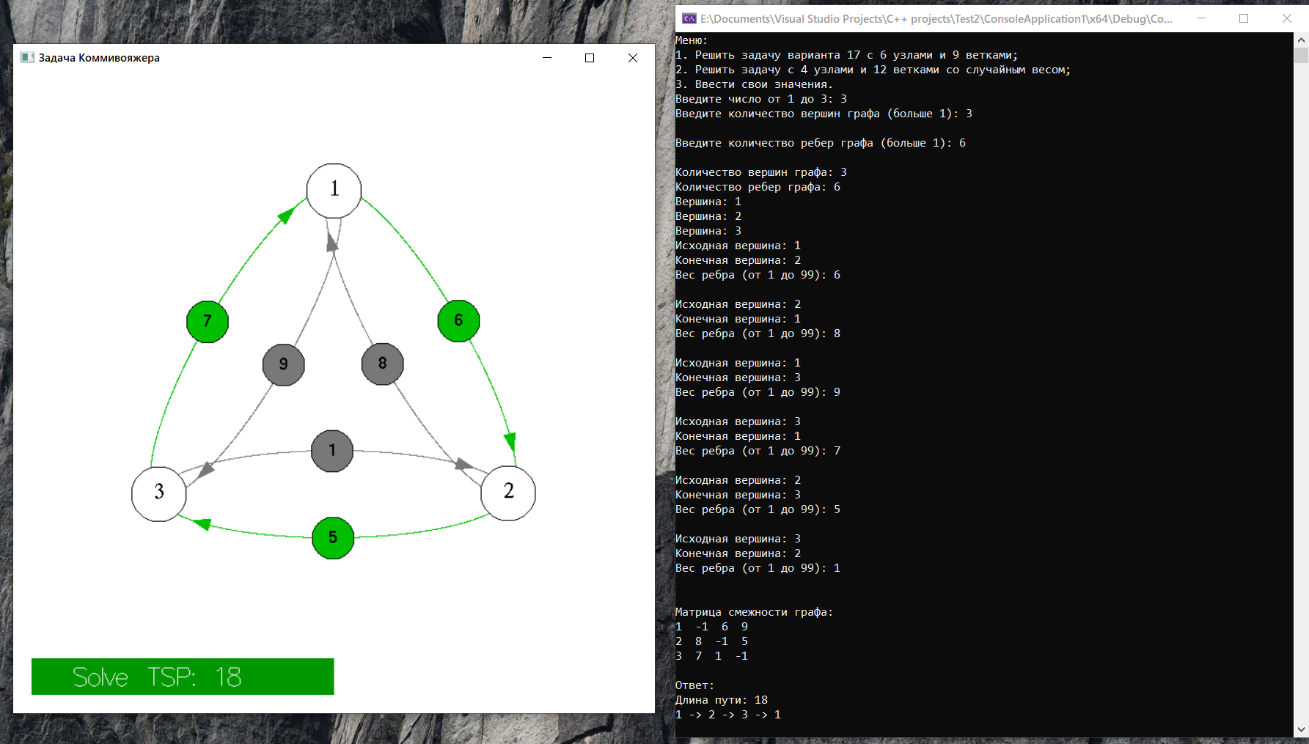
**Тестирование программы**



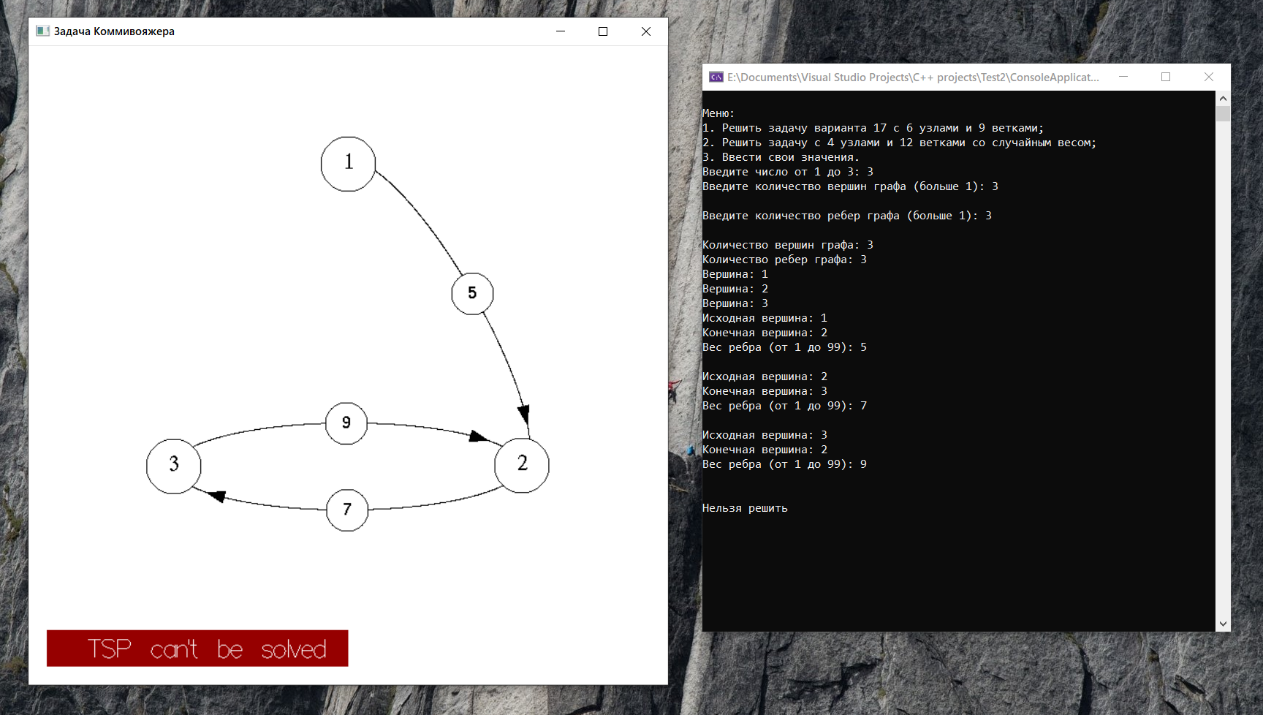
Снимок работы программы с выбором пользователя “1”



Снимок работы программы с выбором пользователя “2”

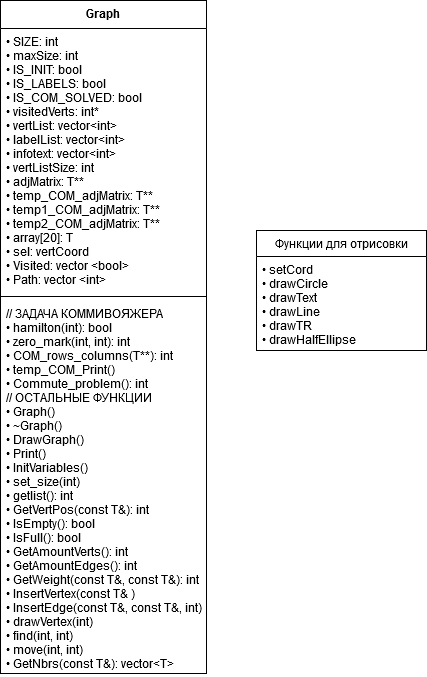
****

Снимок работы программы с выбором пользователя “3”

****

Снимок работы программы с выбором пользователя “3”

**UML-диаграмма**



**Разработка калькулятора**

**Цели работы**

* разработать калькулятор для работы с интервалами времени;
* ввести поддержку ввода данных (в часах, минутах или секундах), операций (нахождения величины временного интервала, суммы, разности) и преобразования интервала в часы, минуты, секунды;
* реализовать алгоритм в виде программы на алгоритмическом языке *С++*;
* разработать интерфейс посредством *Windows Forms*.

**Ход работы**

При работе над калькулятором было уделено внимание минималистичному дизайну интерфейса. Цветовая палитра состоит из спокойных оттенков белого и серого. Текст элементов окрашен в черный цвет, что придает контрастности интерфейсу.

Для сохранения компактности без урезания функционала было решено ограничиться тремя текстовыми полями - по одному для каждой единицы измерения. Они будут получать исходные данных и выводить конечный результат. Эта особенность придает предсказуемость интерфейсу, т.к. пользователь будет знать, где искать нужное ему значение.

Помимо основных функций, указанных в задании, было решено добавить:

* перевод каретки между полями ввода;
* удаление последнего символа в строке;
* очистку всех полей;
* выделение активного поля ввода;
* адаптивное отображение единиц измерения времени;
* ведение истории операций.

**Функции и методы классов**

**Методы класса Graph**

Методы, созданные по умолчанию:

* *MyForm*() – конструктор класса по умолчанию;
* ~*MyFom*() – деструктор класса;
* *InitializeComponent*() – инициализация основных полей класса, кнопок и текстовых полей;

Методы-обработчики событий кнопок:

* *InsertDigit*(*Object*^, *EventArgs*^) – метод-обработчик событий цифровых кнопок (button0 – button9). Вставляет текст кнопки в текущее текстовое поле;
* *BTN\_point\_Click*(*Object*^, *EventArgs*^) – для *BTN\_point*. Переводит каретку из текущего поля в нижнее;
* *BTN\_empty\_Click*(*Object*^, *EventArgs*^) – для *BTN\_delete\_last*. Удаляет последний элемент из текущего текстового поля;
* *BTN\_clear\_Click*(*Object*^, *EventArgs*^) – для *BTN\_clear*. Очищает все текстовые поля и временные переменные;
* *BTN\_add\_Click*(*Object*^, *EventArgs*^) – для *BTN\_add*. Записывает данные из полей в объект класса *clsTime*, очищает поля и задает операция для выполнения;
* *BTN\_minus\_Click*(*Object*^, *EventArgs*^) – для *BTN\_minus*. Записывает данные из полей в объект класса *clsTime*, очищает поля и задает операцию для выполнения;
* *BTN\_equals\_Click*(*Object*^, *EventArgs*^) – для *BTN\_equals*. Записывает данные из полей в объект класса *clsTime*, очищает поля и считает заданную операцию. Вызывает функцию *Display()* с полученным значением;
* *BTN\_to\_hour\_Click*(*Object*^, *EventArgs*^) – для *BTN\_to\_hour*. Записывает данные из полей в объект класса clsTime, очищает поля и переводит заданные данные в часы. Вызывает функцию Display с полученным значением;
* *BTN\_to\_min\_Click*(*Object*^, *EventArgs*^) – для *BTN\_to\_min*. Записывает данные из полей в объект класса *clsTime*, очищает поля и переводит заданные данные в минуты. Вызывает функцию *Display()* с полученным значением;
* *BTN\_to\_sec\_Click*(*Object*^, *EventArgs*^) – для *BTN\_to\_sec*. Записывает данные из полей в объект класса *clsTime*, очищает поля и переводит заданные данные в секунды. Вызывает функцию *Display()* с полученным значением;

Методы-обработчики событий полей:

* *TXT\_main\_h\_TextChanged*(*Object*^, *EventArgs*^)
* *TXT\_main\_m\_TextChanged*(*Object*^, *EventArgs*^)
* *TXT\_main\_s\_TextChanged*(*Object*^, *EventArgs*^)
* *TXT\_got\_focus*(*Object*^, *EventArgs*^)

Прочие методы:

* *ClearAllFields*() – вызывает у всех текстовых полей методы *clear*();
* *Set\_active*(*TextBox*^) – устанавливает фокус и делает активным переданное текстовое поле.
* *TimeSpelling*(*clsTime*^) – подбирает правильное написание для полей переданного объекта *clsTime*. Записывает результат в ярлыки под текстовыми полями: *LBL\_main\_h, LBL\_main\_m, LBL\_main\_s*;
* *Display*(*clsTime*^) – записывает поля объекта класса *clsTime* в текстовые поля;
* *Equals*\_*Display*(*clsTime*^) – записывает поля (включая нули) объекта класса *clsTime* в текстовые поля;
* *GetFields*(*clsTime*^) – записывает данные из текстовых полей в объект класса *clsTime*;
* *PanelColor*() – устанавливает цвет для всех панелей: *PNL\_main\_h,* *PNL\_main\_m, PNL\_main\_s;*
* *InitTXTfocus*() – привязывает события получения и потери фокуса текстовых полей к функции *TXT\_got\_focus*.
* *InitTime*() – создает объекты класса *clsTime*;
* *ResetTime*() – вызывает функцию *reset*() для всех объектов класса *cslTime*;
* *bool* *FieldNotEmpty*() – проверяет все текстовые поля на наличие в них данных. Если все поля пустые, то возвращает значение *false*;

**Методы класса clsTime**

Конструкторы:

* *clsTime*() – конструктор по умолчанию;
* *clsTime(const clsTime^&)* – конструктор копирования;

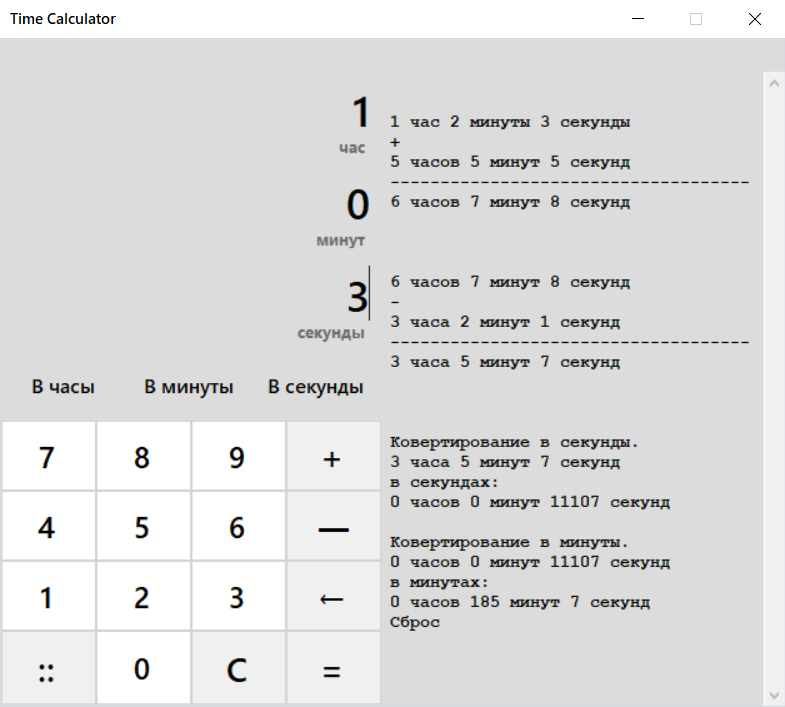
Перегрузка операторов:

* *clsTime*^ *operator+ (const clsTime^)* – перегрузка оператора сложения;
* *clsTime*^ *operator- (const clsTime^)* – перегрузка оператора вычитания;
* *clsTime*^% *operator= (const clsTime%)* – перегрузка оператора присвоения;

Прочие методы:

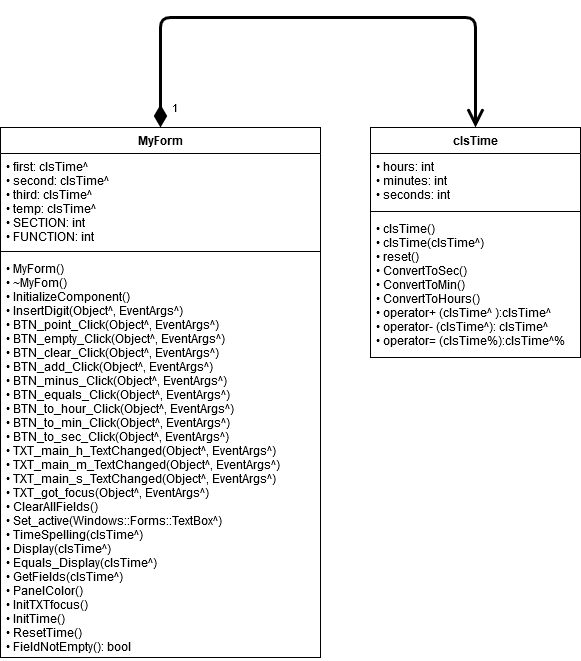
* *void* reset() – присваивает всем полям объекта значение “0”;
* *void* *ConvertToSec*() – приводит все поля объекта к секундам и записывает их сумму в поле *seconds*. Обнуляет поля *minutes* и *hours*;
* *void* *ConvertToMin*() – приводит все поля объекта к минутам и секундам (если значение *seconds* не равно нулю). Обнуляет поле *hours*;
* *void* *ConvertToHours*() – приводит все поля объекта к часам, минутам и секундам (если значения *minutes* и *seconds* не равны нулю);

**Тестирование работы программы**



Снимок работы программы “Калькулятор интервалов времени”

**UML-диаграмма**



**Заключение**

Результатом творческой работы стали две программы.

На языке программирования C++ разработана программа решения задачи коммивояжёра методом ветвей и границ.

На языке программирования C++/CLI был разработан простой калькулятор для работы с интервалами времени.