**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и  программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: аллокаторы

Студент: Юревич В.Ю.

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Создать шаблон динамической коллекцию, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей.
2. В качестве параметра шаблона коллекция должна принимать тип данных - фигуры.
3. Реализовать forward\_iterator по коллекции.
4. Коллекция должны возвращать итераторы begin() и  end().
5. Коллекция должна содержать метод вставки на позицию итератора.
6. Коллекция должна содержать метод удаления из позиции итератора.
7. При выполнении недопустимых операций необходимо генерировать исключения.
8. Итератор должен быть совместим со стандартными алгоритмами.
9. Коллекция должна содержать метод доступа: доступ к элементу по оператору [].
10. Реализовать программу, которая:
    1. позволяет вводить с клавиатуры фигуры и добавлять в коллекцию;
    2. позволяет удалять элемент из коллекции по номеру элемента;
    3. выводит на экран введенные фигуры с помощью std::for\_each;
    4. выводит на экран количество объектов, у которых площадь меньше   заданной (с помощью  std::count\_if).

Вариант 13.

Фигура по варианту: Ромб.

Контейнер по варианту: Список.

Аллокатор:  Вектор.

**Ссылка на репозиторий на GitHub:** https://github.com/vi-yurevich/oop\_exercise\_06

1. **Описание программы**

Для хранения фигуры используется шаблонный класс TRhombus, содержащий координаты центра фигуры и длины диагоналей. Коллекция представлена шаблонным классом TList. Для хранения коллекции используется умный указатель. При добавлении узла выделяется память под элемент типа TRhombus. Реализован класс линейного аллокатора, выделяющего блок фиксированного размера, а затем возвращающего указатели на свободные блоки. Аллокатор такого типа не может освобождать память, поэтому метод deallocate является пустым. Деструктор аллокатора освобождает сразу весь выделенный блок памяти. Для совместимости со стандартными алгоритмами (std::for\_each), были перегружены следующие операторы: \*, ->, ++, ==, !=. Помимо этого реализованы методы Begin() и End(), возвращающие итератор. Реализован метод Insert — осуществляющий вставку элемента в список по итератору, а также метод Erase — осуществляющий удаление элемента из списка по итератору, вдобавок перегружена операция [], для доступа к элементам списка. Элементы добавляются и удаляются из коллекции с помощью итераторов (методов Insert и Erase). Реализован тщательный контроль вводимых данных, а при выполнении недопустимых операций генерируются исключения.

1. **Руководство по использованию программы**

Взаимодействие с пользователем происходит с помощью меню:

1 - Вывести меню

2 - Добавить ромб в список

3 - Печать списка

4 - Вставить по номеру в списке

5 - Удалить по номеру в списке

6 - Вывести количество элементов в списке

7 - Вставить элемент на место головы списка

0 - Выход

Отсчет элементов начинается с нулевой позиции. При вставке по номеру, элемент будет вставляться после номера, который введет пользователь. В случае необходимости изменить голову списка необходимо использовать отдельную функцию.

1. **Набор  тестов**

test\_01.txt:

2 //добавляем фигуру и вводим требуемые данные

0

0

2

4

2 //добавляем фигуру и вводим требуемые данные

1

1

10

5

3 //выводим список

5 //удаляем по номеру

6 //пытаемся удалить элемент с номером превышающим размер списка

5 //удаляем по номеру

1

3 //выводим список

0 //выход из программы

test\_02.txt:

2 //добавляем фигуру в список

-3

-5

-6 //попытка указать отрицательную длину

6

8

4 //вставляем элемент на позицию

7 //попытка вставки элемента по номеру превышающему размер списка

1

1

1

1

4 //вставляем элемент на позицию

0

5

6

7

10

2 //вставляем элемент в конец

10

10

100

1200

3 //печать списка

7 //вставляем новую голову списка

-100

100

8

1

2 //добавляем фигуру в список

3

4

3

6

3 //печатаем список

0 //выход

1. **Результаты выполнения тестов**

test\_01.txt:

./oop\_exercise\_06

1 - Вывести меню

2 - Добавить ромб в список

3 - Печать списка

4 - Вставить по номеру в списке

5 - Удалить по номеру в списке

6 - Вывести количество элементов в списке

7 - Вставить элемент на место головы списка

0 - Выход

Выберете пункт меню: 2

Укажите координату Х центра ромба: 0

Укажите координату Y центра ромба: 0

Укажите длину вертикальной диагонали: 2

Укажите длину горизонтальной диагонали: 4

Выберете пункт меню: 2

Укажите координату Х центра ромба: 1

Укажите координату Y центра ромба: 1

Укажите длину вертикальной диагонали: 10

Укажите длину горизонтальной диагонали: 5

Выберете пункт меню: 3

Координаты вершин: (0;1) (2;0) (0;-1) (-2;0)

Координаты центра: (0, 0)

Площадь ромба: 8

Координаты вершин: (1;6) (3;1) (1;-4) (-1;1)

Координаты центра: (1, 1)

Площадь ромба: 50

Выберете пункт меню: 5

Укажите номер позиции: 6

Ошибка. Попытка удалить элемент на несуществующей позиции списка

Выберете пункт меню: 5

Укажите номер позиции: 1

Выберете пункт меню: 3

Координаты вершин: (0;1) (2;0) (0;-1) (-2;0)

Координаты центра: (0, 0)

Площадь ромба: 8

Выберете пункт меню: 0

test\_02.txt:

./oop\_exercise\_06

1 - Вывести меню

2 - Добавить ромб в список

3 - Печать списка

4 - Вставить по номеру в списке

5 - Удалить по номеру в списке

6 - Вывести количество элементов в списке

7 - Вставить элемент на место головы списка

0 - Выход

Выберете пункт меню: 2

Укажите координату Х центра ромба: -3

Укажите координату Y центра ромба: -5

Укажите длину вертикальной диагонали: -6

Ошибка ввода. Длина не может быть отрицательной. Повторите ввод.

Укажите длину вертикальной диагонали: 6

Укажите длину горизонтальной диагонали: 8

Выберете пункт меню: 4

Укажите номер позиции: 7

Укажите координату Х центра ромба: 1

Укажите координату Y центра ромба: 1

Укажите длину вертикальной диагонали: 1

Укажите длину горизонтальной диагонали: 1

Ошибка. Попытка вставить элемент на несуществующую позицию списка

Выберете пункт меню: 4

Укажите номер позиции: 0

Укажите координату Х центра ромба: 5

Укажите координату Y центра ромба: 6

Укажите длину вертикальной диагонали: 7

Укажите длину горизонтальной диагонали: 10

Выберете пункт меню: 2

Укажите координату Х центра ромба: 10

Укажите координату Y центра ромба: 10

Укажите длину вертикальной диагонали: 100

Укажите длину горизонтальной диагонали: 1200

Выберете пункт меню: 3

Координаты вершин: (-3;-2) (1;-5) (-3;-8) (-7;-5)

Координаты центра: (-3, -5)

Площадь ромба: 48

Координаты вершин: (5;9) (10;6) (5;3) (0;6)

Координаты центра: (5, 6)

Площадь ромба: 70

Координаты вершин: (10;60) (610;10) (10;-40) (-590;10)

Координаты центра: (10, 10)

Площадь ромба: 120000

Выберете пункт меню: 7

Укажите координату Х центра ромба: -100

Укажите координату Y центра ромба: 100

Укажите длину вертикальной диагонали: 8

Укажите длину горизонтальной диагонали: 1

Выберете пункт меню: 2

Укажите координату Х центра ромба: 3

Укажите координату Y центра ромба: 4

Укажите длину вертикальной диагонали: 3

Укажите длину горизонтальной диагонали: 6

Выберете пункт меню: 3

Координаты вершин: (-100;104) (-100;100) (-100;96) (-100;100)

Координаты центра: (-100, 100)

Площадь ромба: 8

Координаты вершин: (-3;-2) (1;-5) (-3;-8) (-7;-5)

Координаты центра: (-3, -5)

Площадь ромба: 48

Координаты вершин: (5;9) (10;6) (5;3) (0;6)

Координаты центра: (5, 6)

Площадь ромба: 70

Координаты вершин: (10;60) (610;10) (10;-40) (-590;10)

Координаты центра: (10, 10)

Площадь ромба: 120000

Координаты вершин: (3;5) (6;4) (3;3) (0;4)

Координаты центра: (3, 4)

Площадь ромба: 18

Выберете пункт меню: 0

1. **Листинг программы**

**main.cpp:**

#include "List.h"

#include "Rhombus.h"

#include "Allocator.h"

#include <iostream>

#include <algorithm>

const std::size\_t BLOCK\_SIZE  = 40;

TRhombus<int, Allocator<int, BLOCK\_SIZE>> EnterRhombus() {

   TRhombus<int, Allocator<int, BLOCK\_SIZE>> rhomb;

   bool key = false;

   do {

       std::cout << "Укажите координату Х центра ромба: ";

       std::cin >> rhomb.center.first;

       if (std::cin.fail()) {

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

           std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

       }

       else {

           key = true;

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

       }

   } while (key != true);

   do {

       std::cout << "Укажите координату Y центра ромба: ";

       std::cin >> rhomb.center.second;

       if (std::cin.fail()) {

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

           std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

       }

       else {

           key = false;

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

       }

   } while (key != false);

   do {

       std::cout << "Укажите длину вертикальной диагонали: ";

       std::cin >> rhomb.vertDiag;

       if (std::cin.fail()) {

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

           std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

       }

       else if (rhomb.vertDiag < 0) {

                   std::cout << "Ошибка ввода. Длина не может быть отрицательной. Повторите ввод.\n";

               }

       else {

           key = true;

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

       }

   } while(key != true);

   do {

       std::cout << "Укажите длину горизонтальной диагонали: ";

       std::cin >> rhomb.horDiag;

       if (std::cin.fail()) {

       std::cin.clear();

       std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

       std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

       }

       else if (rhomb.horDiag < 0) {

                   std::cout << "Ошибка ввода. Длина не может быть отрицательной. Повторите ввод.\n";

               }

       else {

           key = false;

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

       }

   } while(key != false);

   return(rhomb);

}

void EnterPosition (size\_t& pos) {

   bool key = false;

   size\_t i;

   do {

       std::cout << "Укажите номер позиции: ";

       std::cin >> i;

       if (std::cin.fail()) {

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

           std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

       }

       else if (i < 0) {

           std::cout << "Ошибка ввода. Номер позиции в списке может быть только положительным. Повторите ввод.\n";

       }

       else {

           key = true;

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

       }

   } while(key != true);

   pos = i;

}

int main() {

   TRhombus<int, Allocator<int, BLOCK\_SIZE>> rhomb;

   TList<TRhombus<int, Allocator<int, BLOCK\_SIZE>>> list;

   bool key = false;

   int menu = 1;

   int square;

   size\_t i;

   auto Print = [](const TRhombus<int, Allocator<int, BLOCK\_SIZE>>& r) {

       std::cout << r << "\n" << std::endl;

   };

   auto TestSquare = [&square](const TRhombus<int, Allocator<int, BLOCK\_SIZE>>& r) {

       return (RhombusSquare<int, Allocator<int, BLOCK\_SIZE>>(r) < square);

   };

   while(menu != 0) {

       switch(menu) {

           case 1:

               std::cout << "1 - Вывести меню\n2 - Добавить ромб в список\n3 - Печать списка" << std::endl;

               std::cout << "4 - Вставить по номеру в списке\n5 - Удалить по номеру в списке\n6 - Вывести количество элементов в списке" << std::endl;

               std::cout << "7 - Вставить элемент на место головы списка\n0 - Выход" << std::endl;

               break;

           case 2:

               rhomb = EnterRhombus();

               list.PushBack(rhomb);

               break;

           case 3:

               if (list.Size() <= 0) {

                   std::cout << "Ошибка. Сначала необходимо добавить что-нибудь в список" << std::endl;

               }

               else {

                   std::cout << std::endl;

                   std::for\_each(list.Begin(), list.End(), Print);

               }

               break;

           case 4: {

               if (list.Size() <= 0) {

                   std::cout << "Ошибка. Сначала необходимо добавить что-нибудь в список" << std::endl;

               }

               else {

                   EnterPosition(i);

                   rhomb = EnterRhombus();

                   TList<TRhombus<int, Allocator<int, BLOCK\_SIZE>>>::TLIterator iter(list, i);

                   try {

                       list.Insert(iter, rhomb);

                   }

                   catch(std::logic\_error& error){

                       std::cout << error.what();

                   }

               }

           }

           break;

           case 5: {

               if (list.Size() <= 0) {

                   std::cout << "Ошибка. Сначала необходимо добавить что-нибудь в список" << std::endl;

               }

               else {

               EnterPosition(i);

               TList<TRhombus<int, Allocator<int, BLOCK\_SIZE>>>::TLIterator iter(list, i);

               try {

                       list.Erase(iter);

                   }

                   catch(std::logic\_error& error){

                       std::cout << error.what();

                   }

               }

               break;

           }

           case 6:

               std::cout << list.Size() << std::endl;

               break;

           case 7:

           rhomb = EnterRhombus();

           list.InsertToHead(rhomb);

           break;

           case 0:

           break;

       }

       do {

           std::cout << "Выберете пункт меню: ";

           std::cin >> menu;

           if (std::cin.fail()) { //контроль вводимых данных

               std::cin.clear();

               std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

               std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

           }

           else if ((menu < 0) || (menu > 7)) {

               std::cout << "Указанный пункт отсутсвует в меню. Попробуйте ещё раз." << std::endl;

           }

           else {

               key = true;

               std::cin.clear();

               std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

           }

       } while(key != true);

       key = false;

   }

   return(0);

}

**List.h:**

#include <memory>

#include <iostream>

#include <exception>

#include <iterator>

template<class T>

class TList {

   private:

       struct TListNode {

           T data;

           std::shared\_ptr<TListNode> next;

           T& Get(size\_t index) {

               if (index == 0) return(data);

               if (next) return(next->Get(--index));

               throw std::logic\_error("Ошибка. Выход за границу списка\n");

           }

           void PushBack(T& elem) {

               if(next) next->PushBack(elem);

               else next = std::shared\_ptr<TListNode>(new TListNode{elem, std::shared\_ptr<TListNode>()});

           }

           void Insert(TListNode& pred, size\_t index, const T& elem) {

               if (index == 0) {

                   pred.next = std::shared\_ptr<TListNode>(new TListNode{elem, std::move(pred.next)});

               }

               else if (next) {

                   next->Insert(\*this, --index, elem);

               }

               else throw std::logic\_error("Ошибка. Попытка вставить элемент на несуществующую позицию списка\n");

           }

           void Erase(TListNode& pred, size\_t index) {

               if (index == 0) {

                   pred.next = std::move(next);

               }

               else if (next) {

                   next->Erase(\*this, --index);

               }

               else throw std::logic\_error("Ошибка. Попытка удалить элемент на несуществующей позиции списка\n");

           }

           size\_t Size() {

               if (!next) {

                   return 1;

               }

               else return(1 + next->Size());

           }

       };

       std::shared\_ptr<TListNode> head;

   public:

       using value\_type = T;

       class TLIterator {

           private:

               TList& list;

               size\_t index;

               friend class TList;

           public:

               using difference\_type = ptrdiff\_t;

               using value\_type = TList::value\_type;

               using reference = TList::value\_type&;

               using pointer = TList::value\_type\*;

               using iterator\_category = std::forward\_iterator\_tag;

           TLIterator(TList& l, int i) : list(l), index(i) {}

           reference operator\*() {

               return(list[index]);

           }

           pointer operator->() {

               return(&list[index]);

           }

           TLIterator& operator++() {

               ++index;

               return \*this;

           }

           bool operator==(const TLIterator& itr) {

               if (&list == &itr.list) {

                   if (index == itr.index) return(true);

                   else return(false);

               }

               throw std::logic\_error("Ошибка. Попытка сравнения итераторов разных списков\n");

           }

           bool operator!=(const TLIterator& itr) {

               if (&list == &itr.list) {

                   if (index != itr.index) return(true);

                   else return(false);

               }

               throw std::logic\_error("Ошибка. Попытка сравнения итераторов разных списков\n");

           }

       };

       TList() : head(std::shared\_ptr<TListNode>()) {}

       size\_t Size() {

           if (head) {

               return (head->Size());

           }

           else return((size\_t)0);

       }

       value\_type& operator[](size\_t i) {

           if (!head) throw std::logic\_error("Ошибка. Попытка доступа к несуществующему элементу\n");

           return(head->Get(i));

       }

       void PushBack(value\_type& elem) {

           if (!head) {

               head = std::shared\_ptr<TListNode>(new TListNode{elem, std::shared\_ptr<TListNode>()});

           }

           else {

               head->PushBack(elem);

           }

       }

       void PushBack(value\_type&& elem) {

           if (!head) {

               head = std::shared\_ptr<TListNode>(new TListNode{elem, std::shared\_ptr<TListNode>()});

           }

           else {

               head->PushBack(elem);

           }

       }

       TLIterator Erase(TLIterator itr) {

           if (itr.index == 0) {

               if (head) {

               head = std::move(head->next);

               }

               else throw std::logic\_error("Ошибка. Попытка удалить элемент на несуществующей позиции списка\n");

           }

           else if (head->next) {

               try {

                   head->next->Erase(\*head, --itr.index);

               }

               catch(std::logic\_error& error) {

                   throw error;

               }

           }

           else throw std::logic\_error("Ошибка. Попытка удалить элемент на несуществующей позиции списка\n");

           if (itr.index < Size()) return(itr);

           else return(TLIterator(\*this, Size()));

       }

       TLIterator Insert(TLIterator itr, const value\_type& elem) {

           if (itr.index == 0) {

               if (head) {

                   head->Insert(\*head, itr.index, elem);

               }

               else throw std::logic\_error("Ошибка. Попытка вставить элемент на несуществующую позицию списка\n");

           }

           else if (head->next) {

               try {

                   head->next->Insert(\*head, --itr.index, elem);

               }

               catch(std::logic\_error& error) {

                   throw error;

               }

           }

           else throw std::logic\_error("Ошибка. Попытка вставить элемент на несуществующую позицию списка\n");

           return(itr);

       }

       void InsertToHead(const value\_type& elem) {

           head = std::shared\_ptr<TListNode>(new TListNode{elem, std::move(head)});

       }

       TLIterator Begin() {

           return(TLIterator(\*this, 0));

       }

       TLIterator End() {

           return(TLIterator(\*this, Size()));

       }

};

**Rhombus.h:**

#include <iostream>

#include <utility>

template<class T, class ALLOCATOR>

struct TRhombus {

   std::pair<T, T> center;

   T vertDiag;

   T horDiag;

   TRhombus() : TRhombus(std::pair<T, T>{0, 0}, 0, 0) {}

   TRhombus(const std::pair<T, T>& c, const T& v, const T& h) : center(c), vertDiag(v), horDiag(h) {}

   TRhombus(const TRhombus& r) {

       center = r.center;

       vertDiag = r.vertDiag;

       horDiag = r.horDiag;

   }

   TRhombus& operator=(const TRhombus& r) {

       center = r.center;

       vertDiag = r.vertDiag;

       horDiag = r.horDiag;

       return(\*this);

   }

   using allocator\_type = typename ALLOCATOR::template rebind<TRhombus<T, ALLOCATOR>>::other;

           static allocator\_type &get\_allocator() {

               static allocator\_type allocator;

               return allocator;

           }

           void \* operator new(std::size\_t n) {

               return get\_allocator().allocate(sizeof(TRhombus<T, ALLOCATOR>));

           }

           void operator delete(void \* p) {

               get\_allocator().destroy((TRhombus<T, ALLOCATOR>\*)p);

               get\_allocator().deallocate((TRhombus<T, ALLOCATOR> \*)p, sizeof(TRhombus<T, ALLOCATOR>));

           }

};

template<class T, class ALLOCATOR>

T RhombusSquare (const TRhombus<T, ALLOCATOR>& ob) {

   return(ob.vertDiag \* ob.horDiag);

}

template<class T, class ALLOCATOR>

std::ostream& operator<<(std::ostream& os, const TRhombus<T, ALLOCATOR>& rhombus) {

   os << "Координаты вершин: ";

   os << "(" << rhombus.center.first << ";" << (rhombus.center.second + rhombus.vertDiag / 2) << ") ";

   os << "(" << (rhombus.center.first + rhombus.horDiag / 2)  << ";" << rhombus.center.second << ") ";

   os << "(" << rhombus.center.first << ";" << (rhombus.center.second - rhombus.vertDiag / 2) << ") ";

   os << "(" << (rhombus.center.first - rhombus.horDiag / 2) << ";" << rhombus.center.second << ")";

   os << "\n";

   os << "Координаты центра: " << rhombus.center << "\n";

   os << "Площадь ромба: " << RhombusSquare<T>(rhombus) << "\n";

   return(os);

}

template<class T1, class T2>

std::ostream& operator<<(std::ostream& os, const std::pair<T1, T2>& p) {

   os << "(" << p.first << ", " << p.second << ")";

   return os;

}

**Allocator.h:**

#include <stack>

#include <iostream>

#include <vector>

template<class T, std::size\_t BLOCK\_SIZE>

class Allocator {

   private:

       std::vector<T\*> vec;

       T\* buffer;

       size\_t countUsed;

   public:

   using allocator\_type = Allocator;

   using value\_type = T;

   using pointer = T\*;

   using reference = T&;

   using const\_reference = const T&;

   using size\_type = std::size\_t;

   Allocator(): countUsed(0), buffer(nullptr) {

       static\_assert(BLOCK\_SIZE > 0, " ");

   }

   ~Allocator() {

       delete[] buffer;

   }

   template<class U>

   struct rebind {

       using other = Allocator<U, BLOCK\_SIZE>;

   };

   T\* allocate(std::size\_t n) {

       if (buffer == nullptr) {

           buffer = new T [BLOCK\_SIZE];

           for (std::size\_t i = 0; i < BLOCK\_SIZE; ++i){

               vec.push\_back(&buffer[i]);

           }

       }

       else if ((vec.size() - countUsed) < n) {

           std::bad\_alloc err;

           throw err;

       }

       T\* res = nullptr;

       res = vec[countUsed];

       countUsed += n;

       return (res);

   }

   void deallocate (T\* p, std::size\_t ) {}

   void destroy(T\* p) {

       p->~T();

   }

   template<typename U, typename... Args>

   void construct(U \* p, Args&&...args) {

       new (p) U(std::forward<Args>(args)...);

   }

};

1. **Выводы**

    В процессе выполнения данной лабораторной работы я получил навыки работы с аллокаторами памяти в Си++. Реализовал собственный аллокатор, узнал как перегружать операторы выделения и освобождения памяти, используемые в моём списке. Реализованный мною аллокатор является совместимым с контейнерами из STL.

**Список литературы**

1. Про аллокаторы в С++ [Электронный ресурс]. URL:<https://habr.com/ru/post/505632/>(дата обращения: 04.12.2020).
2. [cppreference.com](https://en.cppreference.com/) [Электронный ресурс]. URL: <https://en.cppreference.com/w/> (дата обращения: 06.12.20).